IBM App Connect Enterprise
12.0

IBM App Connect Enterprise 12.0; User Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 3077.

Contents

Chapter 1. Planning your IBM App Connect Enterprise solution

Use the topics in this section to help you plan your implementation of IBM® App Connect Enterprise.

Finding the latest information

Access the latest information for IBM App Connect Enterprise, including system requirements and product documentation.

About this task

The following information is provided:

System requirements web page
For the latest details of hardware requirements, operating systems, and software requirements on all supported platforms, visit the IBM App Connect Enterprise system requirements web page.

`readme.html` file
The product `readme.html` file is frequently updated and includes information about last-minute changes and known problems and work-arounds. You can find the current readme file at https://www.ibm.com/support/pages/node/318359.

Supporting programs for web page
For a list of supporting program versions for IBM App Connect Enterprise 12.0, IBM App Connect Enterprise for Developers 12.0, and IBM App Connect Standard 12.0, see Supporting programs for IBM App Connect.

Online documentation
The main source of product documentation is online at https://www.ibm.com/docs/app-connect/12.0, which is also accessible from the IBM App Connect Enterprise Toolkit with an internet connection.

Understanding your data and message processing requirements

It is important to be clear about your processing requirements, and to understand what you want IBM App Connect Enterprise to do for your business.

About this task

In a typical IT environment there are many interacting components, each with a role to play, and it is important that you are able to clearly define the role of IBM App Connect Enterprise in the broader environment. A key design decision is how much business logic to implement within your message flows. IBM App Connect Enterprise enables you to implement large amounts of processing within a message flow, and implementations can vary significantly from the simple routing of messages to complex validation and transformation. Some implementations also read data from a database and use that data to populate messages. Regardless of the amount of function that you decide to implement in a message flow, it is important to have clear boundaries between pieces of application processing.

Several different processing styles are commonly used with IBM App Connect Enterprise, and it is important to understand the ones that are most relevant to you, such as:

Request Reply
This is the most common type of processing, and enables two applications to communicate, even if they use different data formats. For example, a message flow transforms a request message from Application 1 into a format that Application 2 can understand. The output of the first message flow is then sent to Application 2, which processes the request and issues a reply. The reply is processed in a second message flow, which converts the response message into a format that Application 1 can understand.

Aggregation
This type of processing is often used to invoke one or more back-end systems and coordinate replies. It is a more complex form of a Request Reply case, in which all the replies from the intermediate applications must be collected together before the reply message for the original request can be sent. For example, this type of processing could be used to book a holiday, in which a flight, hotel, and a car are required, and all must be successfully processed before the holiday confirmation can be sent.

Routing
Routing is used to redirect messages, and one or more copies of a message can be sent to one or more destinations.

Transformation
This type of processing involves the use of one or more transformation technologies such as Compute, JavaCompute, XMLT, or Mapping node. In this type of processing the input message is processed according to some business rules, and there might also be a change of message format or protocol.

File processing
The use of files is one of the most common methods of storing data. You can create message flows to process data in files, accepting data in files as input message data, and producing output message data for file-based destinations.

Database handling
You can configure your message flows to access and manipulate business data in databases.

Web services
IBM App Connect Enterprise can be used as both a consumer and provider of web services.

Transport switching
You can use this type of processing to switch between transports such as HTTP and JMS.

For information about how to choose the appropriate edition of IBM App Connect Enterprise to suit your requirements, see "Choosing your IBM App Connect Enterprise edition and operation mode" on page 2. For more information about the capabilities provided by IBM App Connect Enterprise, see features.

Choosing your IBM App Connect Enterprise edition and operation mode

The capability and capacity provided by IBM App Connect Enterprise vary according to the operation mode in which your integration servers are running, and your entitlement to run in a particular mode depends on the edition of the product that you have purchased.

About this task

You can choose the mode of operation based on the functionality and capacity that you require. You must ensure that your use and configuration of the product conforms to the license agreement that you have purchased. Some features of IBM App Connect Enterprise have specific requirements, such as the availability of an additional product, such as a database.

The following topics describe the features that are available, together with additional requirements and restrictions:

- features
- "Operation modes" on page 3
- "Restrictions that apply in each operation mode" on page 5

For more information about the license requirements for each operation mode, see "License requirements" on page 5.

Operation modes

The operation mode that you can use for your integration servers is determined by the license that you purchase.

The following modes are supported:

- "Advanced mode" on page 3. All features are enabled and no restrictions or limits are imposed. This mode is the default mode, unless you have the Developer Edition.
- "Nonproduction mode" on page 3. All features are enabled and no functional limits are imposed, but you can use the product for evaluation, development, and test purposes only.
- "Nonproductionfree mode" on page 4. All features are enabled and no functional limits are imposed, but you can use the product for development and functional testing purposes only (not for performance, scalability, or security testing).
- "Developer mode" on page 4. All features are enabled, but you can use the product for evaluation, development, and test purposes only. Developer Edition is limited to one message (transaction) per second at the message flow level.
- "Standard mode" on page 4. All features are enabled. If you are using an integration node, you can associate only one integration server with it, but the number of message flows that you can deploy to it is unlimited.

Operation modes typically restrict the number of message flow nodes that are available, and integration server capacity, but many features are available across all modes of operation. For an overview of the main IBM App Connect Enterprise capabilities, see features.

You must ensure that your integration servers are running in the operation mode for which you have purchased a license.

If you have purchased a license for the full package or the Standard Edition, your integration servers are automatically created in `advanced` mode unless you specify the correct mode for your license.

Change the mode of your integration node to conform to your license if necessary; see "Changing the operation mode" on page 86. You can also report the current mode of your integration node; see "Checking the operation mode" on page 87.

The IBM App Connect Enterprise Toolkit remains the same in all modes. All the capabilities of the IBM App Connect Enterprise Toolkit are available in all modes. If you try to deploy too many message flows or integration servers for the mode, or try to use a message flow node that is not valid in the mode, the operation is rejected, and an error message is displayed indicating the reason for the failure; see "Restrictions that apply in each operation mode" on page 5. Restrictions for a given mode also apply to the use of message flows generated by IBM App Connect Enterprise patterns.

If you intend to use the IBM App Connect Enterprise features that require MQ functionality, you can install and use IBM MQ within the terms of your IBM App Connect Enterprise license. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

Advanced mode

In `advanced` mode, all features are enabled, and there are no operational limits on the creation of integration servers or on the number of flows that are deployed to an individual integration server. If you want to use all or most of the features available, your integration servers must operate in this mode, and you therefore require the full license. If you do not specify another mode, your integration servers have the mode set to the default value, which is `advanced`.

Nonproduction mode

In `nonproduction` mode, all features are enabled. You can use all available function, and no limits are imposed on the number of resources that you can create or the rate at which messages can be processed. The functions that are available in `nonproduction` mode are the same as in `advanced` mode, but `nonproduction` mode can be used for evaluation, development, and test purposes only. When you are

ready to move to a production environment, you will need to obtain the full IBM App Connect Enterprise license and change your integration servers to `advanced` mode.

Nonproductionfree mode

In `nonproductionfree` mode, all features are enabled. You can use all available function, and no limits are imposed on the number of resources that you can create or the rate at which messages can be processed. The functions that are available in `nonproductionfree` mode are the same as in `advanced` mode, but `nonproductionfree` mode can be used for development and functional testing purposes only. For non-functional testing (including performance, scalability, and security testing), you must use `nonproduction` mode and obtain a license for that mode. When you are ready to move to a production environment, you will need to obtain the full IBM App Connect Enterprise license and change your integration servers to `advanced` mode. For more information about license agreements, contact your IBM representative.

Developer mode

In `developer` mode, all features are enabled. You can use all available function, and you are not limited in the number of resources that you can create and maintain. IBM App Connect Enterprise for Developers (Developer Edition) is provided for evaluation, development, and test purposes only.

Developer Edition is limited to one message (transaction) per second at the message flow level. Each message flow is able to process one message per second, irrespective of the number of input nodes that are attached to the message flow, or the number of additional instances. When using Developer Edition, if a policy is deployed to increase the message rate throughput above one message per second, a message is reported in the syslog and event log stating that the message rate cannot be changed in Developer Edition. The policy that is deployed has no effect on the message rate.

You can download Developer Edition at no charge from https://www.ibm.com/marketing/iwm/iwm/web/dispatcher.do?source=swg-wmbfd, and you are free to use it for as long as you require, within the terms of the license. There is no expiry period for the Developer Edition license.

When you have installed Developer Edition, if you subsequently purchase a license and install the full version of IBM App Connect Enterprise, any Developer mode integration servers that are started with the full version are treated as Advanced mode integration servers. In this case, you must also modify the operation mode of these integration servers by using the **`mqsimode`** command to reflect the license that you have purchased. For more information, see “Changing the operation mode” on page 86.

Standard mode

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

You can use all the available functions, but are limited in the number of resources that you can create and maintain. You can associate only one integration server with an integration node; for more information, see “Restrictions that apply in each operation mode” on page 5. If you attempt to exceed the limits of this mode, the deployment is rejected.

Development and unit test

Your license also covers use of the product for development and unit test purposes, but check the license to ensure that you conform to any restrictions for development and unit test. You can view the license for IBM App Connect Enterprise by visiting the Software license agreements search website. Search for "IBM App Connect Enterprise" and choose the license that applies to the version you are using.

Contact your IBM representative if you want further details about license agreements, or if you want to purchase additional licenses or change the type of license that you have purchased.

Integration with Tivoli License Manager

If you use IBM Tivoli® License Manager to control and manage your licensed software products, you must ensure that you choose the correct license for the IBM App Connect Enterprise edition that you have purchased. For more information, see “Installing IBM License Metric Tool” on page 100.

Restrictions that apply in each operation mode

The operation mode in which you are running your installation of IBM App Connect Enterprise determines how many integration servers you can use, and the purposes for which you can use the product.

Advanced mode

All features are enabled, and there is no limit to the number of integration servers or the number of message flows that can be deployed to each integration server.

For more information about the features that are available in IBM App Connect Enterprise, see features.

Nonproduction mode

All features are enabled, and no limits are imposed on the number of resources that you can create or the rate at which messages can be processed. The functions that are available in `nonproduction` mode are the same as in `advanced` mode, but `nonproduction` mode can be used for evaluation, development, and test purposes only.

Nonproductionfree mode

All features are enabled, and no limits are imposed on the number of resources that you can create or the rate at which messages can be processed. The functions that are available in `nonproductionfree` mode are the same as in `advanced` mode, but `nonproductionfree` mode can be used for development and functional testing purposes only. For non-functional testing (including performance, scalability, and security testing), you must use `nonproduction` mode and obtain a license for that mode.

Developer mode

All features are enabled for an unlimited time, but you can use the product for evaluation purposes only (within the terms of the license). IBM App Connect Enterprise for Developers, which is also known as Developer Edition, is restricted to processing one message (transaction) per second. For more information, see “Operation modes” on page 3.

Standard mode

All features are enabled. If you are using an integration node, you can associate only one integration server with it, but the number of message flows that you can deploy to it is unlimited.

License requirements

You must ensure that your use and configuration of the product conforms to the license agreement that you have purchased.

The following list shows the IBM App Connect Enterprise operation modes that can be used with each of the IBM App Connect Enterprise license agreements:

IBM App Connect Enterprise for Developers (Developer Edition)
`developer` mode

IBM App Connect Enterprise Standard Edition
`standard` mode

IBM App Connect Enterprise (unrestricted license)
`advanced` mode

`developer` mode

`standard` mode

For more information, see “Operation modes” on page 3.

If you choose to change your license agreement from a full license to one of the specialized licenses, you might find that your current configuration is no longer supported. For further details about what features are available for each license, and how to configure your environment, see technical overview.

You can upgrade to the full license from another edition, if appropriate, by purchasing another license.

When you purchase a license for IBM App Connect Enterprise, your license entitles you to install and use IBM MQ, which enables you to use the IBM App Connect Enterprise features that require MQ functionality. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

You can view the full license for IBM App Connect Enterprise by visiting the Software license agreements search website. Search for "IBM App Connect Enterprise" and choose the license that applies to the version you are using.

You can view licenses after installation in your chosen language in directory *install_dir*`/server/license/`. Terms and conditions are also supplied for third-party products that are used by IBM App Connect Enterprise. The file containing these details is stored in the same license subdirectory when you install one or more runtime components.

Contact your IBM representative if you want further details about license agreements, or if you want to purchase additional licenses or change the type of license that you have purchased.

Installing IBM License Service and adding annotations for IBM App Connect Enterprise containers

If you create your own container with IBM App Connect Enterprise, you must install an IBM License Service operator and add annotations to your container to ensure that you comply with the IBM licensing requirements.

License Service is required for monitoring and measuring license usage of IBM App Connect Enterprise in accordance with the pricing rule for containerized environments. You must deploy License Service on all clusters where IBM App Connect Enterprise is installed. Manual license measurements are not allowed. License annotations let you track usage based on the limits defined on the container, rather than on the underlying machine. You configure your clients to deploy the container with specific annotations that the IBM License Service then uses to track usage.

The integrated licensing solution collects and stores the license usage information, which can be used for audit purposes and for tracking license consumption in cloud environments. The solution works in the background and does not require any configuration. Only one instance of the License Service is deployed per cluster regardless of the number of Cloud Paks and containerized products that you have installed on the cluster.

You must deploy License Service on each cluster where IBM App Connect Enterprise is installed. License Service can be deployed on any Kubernetes cluster. For more information about License Service, including how to install and use it, see the License Service documentation.

To ensure license reporting continuity for license compliance purposes, ensure that License Service is successfully deployed and periodically verify whether it is active. For example, to validate whether License Service is deployed and running on the cluster, you can log in to the cluster and run the following command:

```
`kubectl get pods --all-namespaces | grep ibm-licensing | grep -v operator`
```

The following response is a confirmation of successful deployment:

```
1/1     Running
```

For more information about the supported environments and installation instructions, see the ibm-licensing-operator page on GitHub. For further information about IBM container licenses, see the IBM Container Licenses page on Passport Advantage.

IBM License Service is installed on the Kubernetes cluster where the IBM App Connect Enterprise container is deployed, and pod annotations are used to track usage. Clients must be configured to deploy the pod with specific annotations that the IBM License Service uses. Based on your entitlement and capabilities deployed within the container, use one or more of the following annotations:

- "IBM App Connect Enterprise" on page 7
- "IBM App Connect Enterprise for non-production" on page 7
- "IBM App Connect Enterprise for Developers" on page 8

Each of your annotations must contain a single product metric.

IBM App Connect Enterprise

Use one of the following annotations for IBM App Connect Enterprise to specify your chosen product metric, which can be either VIRTUAL_PROCESSOR_CORE or PROCESSOR_VALUE_UNIT, but not both:

VPC:

```
productName: "IBM App Connect Enterprise"
productID: "b8b6252aa88b4cd996c4b7aca350d2fe"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers : "All"
```

PVU:

```
productName: "IBM App Connect Enterprise"
productID: "b8b6252aa88b4cd996c4b7aca350d2fe"
productMetric: "PROCESSOR_VALUE_UNIT"
productChargedContainers : "All"
```

IBM App Connect Enterprise for non-production

Use one of the following annotations for IBM App Connect Enterprise for non-production to specify your chosen product metric, which can be VIRTUAL_PROCESSOR_CORE, PROCESSOR_VALUE_UNIT, or FREE:

VPC:

```
productName: "IBM App Connect Enterprise for non-production"
productID: "eb5b5e73f62b4dcf8c434c6274a158a7"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers : "All"
```

PVU:

```
productName: "IBM App Connect Enterprise for non-production"
productID: "eb5b5e73f62b4dcf8c434c6274a158a7"
productMetric: "PROCESSOR_VALUE_UNIT"
productChargedContainers : "All"
```

FREE:

```
productName: "IBM App Connect Enterprise for non-production"
productID: "eb5b5e73f62b4dcf8c434c6274a158a7"
productMetric: "FREE"
```

You cannot specify multiple product metrics in the same annotation.

IBM App Connect Enterprise for Developers

Use the following annotation for IBM App Connect Enterprise for Developers:

```
productName: "IBM App Connect Enterprise for Developers - FREE"
productID: "53ec37b661cb40e693639822785f02f2"
productMetric: "FREE"
```

Planning an IBM App Connect Enterprise environment

When you start to plan your IBM App Connect Enterprise environment, you must first consider the design of your system infrastructure. A major consideration when you are planning to use IBM App Connect Enterprise is how it will fit into your overall system architecture, and it is important to understand the movement of data around your whole system.

About this task

It is also important to understand the high-level goals of the system that you are building, and to be clear about the functional requirements. For example, IBM App Connect Enterprise has extremely powerful routing and transformation capabilities, but it is not designed to function as an application server whose primary function is data processing that involves a high number of complex calculations.

Another important consideration is whether you require a highly available (HA) system that can withstand the failure of an individual integration server or integration node. For more information about high availability, see “Configuring integration nodes for high availability” on page 120.

The capability and capacity of your integration nodes is partly determined by the operation mode in which the integration nodes are running, and this is determined by the IBM App Connect Enterprise license that you have purchased. For more information, see features and “Operation modes” on page 3.

The following topics describe these factors.

Planning the high-level infrastructure

When you are planning to use IBM App Connect Enterprise, consider your complete system of products as a whole and ensure that the flow of data throughout the system is carefully planned.

About this task

It is important to understand how IBM App Connect Enterprise will fit into the overall system of products in your business, and to be clear about the movement of data around the whole system. It is also important to understand the high-level goals of the system and your functional requirements, so that you can ensure that you are planning to use IBM App Connect Enterprise in a way that will gain the maximum benefit. For example, IBM App Connect Enterprise has powerful routing and transformation capabilities, but it is not designed to function as an application server whose primary function is data processing with a high number of complex calculations. Therefore, it is important to fully understand the role of the product within your overall IT system, in order to use it to its full advantage.

Some common considerations are listed here to assist with your planning.

- Consider whether you need to install IBM MQ to access more IBM App Connect Enterprise functions. For example, if you want global transaction management in your solution. For more information about what functions require access to IBM MQ, see “IBM App Connect Enterprise features requiring supplementary products” on page 9.
- If you are planning to use databases, check the database documentation for information about connections, and the limits or restrictions that might apply. For example, there might be a limit to the maximum number of connections that are allowed at any one time.

The following topics describe some additional factors to consider when you plan your implementation of IBM App Connect Enterprise:

IBM App Connect Enterprise features requiring supplementary products

IBM App Connect Enterprise has no mandatory prerequisites; however, a subset of capabilities require access to other products.

These features affect the main IBM App Connect Enterprise product operations, not your applications. For information about how these products integrate with IBM App Connect Enterprise, see technical overview. For more information about IBM App Connect Enterprise features that have specific requirements, such as a particular mode of operation, product licenses, or access to additional products, see features.

The following table shows the products and systems that are required for specific capabilities in IBM App Connect Enterprise:

Capabilities	Required system
IBM MQ capabilities: • MQ nodes • MQ FTE nodes • CD nodes • Sequence nodes • Timeout nodes • Aggregate nodes • Collector nodes • SAP nodes (if transactional support is required in the flow) • Record-Replay capability • Global transaction support • HTTP proxy servlet • Multi-instance integration nodes	IBM MQ
Database capabilities: Database nodes	For a full list of compatible databases, review the *Detailed System Requirements* in the IBM App Connect Enterprise system requirements for your operating system.
Security profiles	Any of the following systems or protocols: Lightweight Directory Access Protocol (LDAP) Tivoli Federated Identity Manager (TFIM) Microsoft Active Directory (MSAD)

Review the following sections to learn what is required to use these capabilities.

IBM MQ

On distributed systems, IBM MQ is not provided as part of the IBM App Connect Enterprise installation package; however, when you purchase a license for IBM App Connect Enterprise, your license entitles you to install and use IBM MQ. If you choose to use the IBM App Connect Enterprise features that require MQ functionality, you can install and use IBM MQ within the terms of the license. You can install IBM MQ at any time.

The following functions require access to system queues on a queue manager that is specified on the integration node:

- Sequence nodes
- Timeout nodes
- Aggregate nodes
- Collector nodes
- Record-Replay capability
- CD nodes
- HTTP proxy

If you want to use these functions, you must specify a queue manager for the integration node, and then explicitly define a set of MQ objects for that queue manager. For more information, see command.

For information about the supported versions of IBM MQ, see the *Detailed System Requirements* in the IBM App Connect Enterprise system requirements for your operating system.

Database capabilities

You can connect a number of databases with IBM App Connect Enterprise. To learn more about databases, see "Databases overview" on page 1131.

Security profiles

Security profiles form part of optional message flow security. To learn more about security profiles, see "Security profiles" on page 2552.

Naming conventions for integration nodes and associated resources

Establish a naming convention for all the IBM App Connect Enterprise resources in your integration node environment to ensure that names are unique, and that users that create new resources can be confident of not introducing duplication or confusion.

Consider the names that you use for your integration nodes and resources:

Integration nodes

When you create an integration node, give it a name that is unique within your integration node environment. You must use the same name for that integration node when you create it on the system in which it is installed, by using the command **`mqsicreatebroker`**. Integration node names are case-sensitive, except on Windows platforms.

Databases

Ensure that the databases that you use for application data, which are accessed through your deployed message flows, are uniquely named throughout your network, so that confusion and errors are avoided by all your users.

Integration servers

Each integration server name must be unique within an integration node.

Message flows and message processing nodes

Each message processing node must be unique within the message flow it is assigned to. Nodes within the same message flow must not share the same name, even if they are of different types. For example, if you include two MQOutput nodes in a single message flow, provide a unique name for each one.

Message flow names must be unique within the integration node. All references to that name are always to the same message flow. If you assign the same message flow to more than one integration node, you must ensure that you maintain unique message flow names across those integration nodes.

Message sets and messages

Each message name must be unique within the message set to which it belongs.

Message set names must be unique within the integration node. All references to that name are always to the same message set. If you assign the same message set to more than one integration node, you must ensure that you maintain unique message set names across those integration nodes.

IBM MQ objects

If you are using IBM MQ services and objects with your IBM App Connect Enterprise resources, you must consider what conventions to adopt for IBM MQ object names. If you already have an IBM MQ naming convention, use a compatible extension of this convention for IBM App Connect Enterprise resources.

Ensure that every queue manager name is unique within your network of interconnected queue managers, even if some queue managers are not in your IBM App Connect Enterprise network. Unique naming ensures that each queue manager can unambiguously identify the target queue manager that a message must be sent to, and that IBM App Connect Enterprise applications can also interact with IBM MQ applications.

IBM MQ supports a number of objects that are defined to queue managers. These objects (queues, channels, and processes) also have naming conventions and restrictions. For more information about IBM MQ naming conventions, see the topic *Naming IBM MQ objects* in the IBM MQ product documentation.

Business planning

Consider the role of IBM App Connect Enterprise in your overall business solution, and understand the potential benefits and limitations based on your functional and business requirements.

About this task

It is important to understand the capability and capacity that you require from your system. The capability and capacity of your integration nodes are partly determined by the operation mode in which they are running, and this is determined by the IBM App Connect Enterprise license that you have purchased. For information about the features and modes of operation provided in each edition, see “Choosing your IBM App Connect Enterprise edition and operation mode” on page 2.

It is also important to consider the most suitable technology for your business; this varies between projects and organizations, and there are various factors to consider when assessing your requirements:

- The pool of development skills available. For example, if your developers are already skilled in the use of ESQL, you might want to continue using this transformation technology. However, if you have strong Java™ skills, the JavaCompute node might be the better choice, rather than having to teach your Java developers ESQL.
- The ease with which you can teach developers a new development language. For example, if you have many developers it might be not be viable to teach all of them ESQL, and you might choose instead to educate only a few people who can then develop common functions or procedures.
- The skills of individual people. If your developers have relatively little experience with IBM App Connect Enterprise, you might find that the graphical interface provided by the Mapping node is more suitable than other transformation technologies.
- Asset reuse. If you are an existing user of IBM App Connect Enterprise, you might already have large amounts of ESQL code that you would like to reuse. You might also have other resources that you want to reuse, such as Java classes for key business processing, style sheets, or mappings. You might want to use these as the core of message flow processing and extend the processing with one of the other technologies.
- Performance. If message throughput is a prime consideration, you might choose to use ESQL or Java. For more information about designing your system for optimum performance, see “Message flow design and performance” on page 2689 and “Code design and performance” on page 2690.

- Solution scalability, cost and flexibility. You might have an existing network infrastructure that you want to use, or you might want to take advantage of IBM App Connect Enterprise cloud capabilities. For more information on choosing a solution for your installation, see “Cloud overview” on page 12.

For more information about factors to consider when planning your IBM App Connect Enterprise solution, see the following topics:

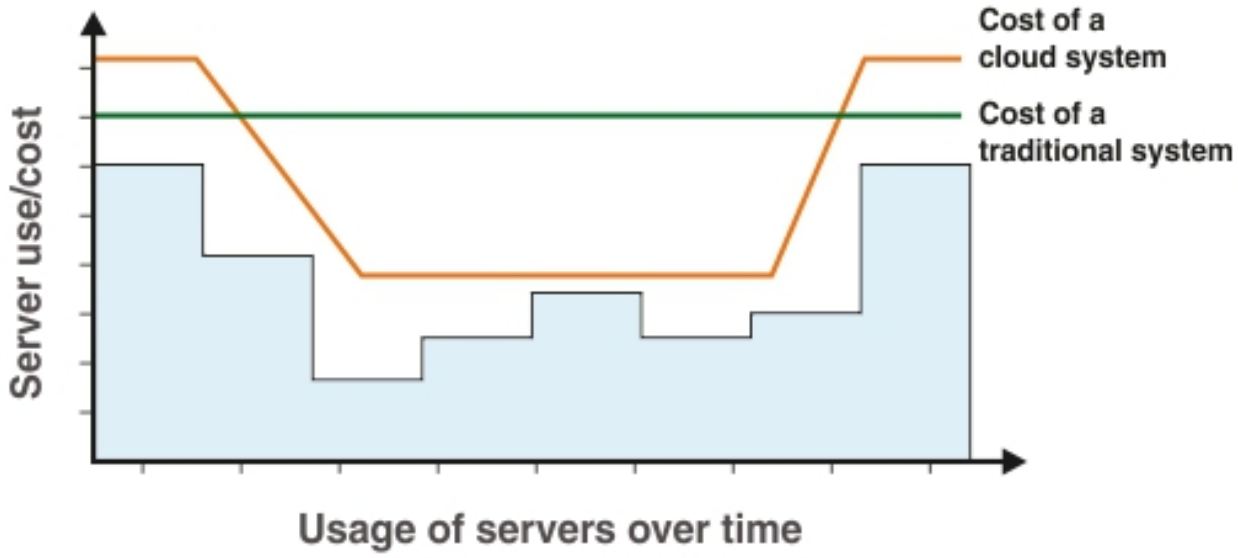
- “Planning an IBM App Connect Enterprise environment” on page 8
- “Planning the high-level infrastructure” on page 8
- “Understanding your data and message processing requirements” on page 1
- “Performance planning” on page 16
- “Planning for security” on page 17

Cloud overview

When you are deciding on your environment for IBM App Connect Enterprise, consider your requirements and how you might want to implement a cloud solution.

Depending on your requirements, you might benefit from reduced operating costs and deployment flexibility for IBM App Connect Enterprise when compared to a static installation.

Using a cloud vendor usually has a higher operating cost than using a static network-based solution, but less upfront capital is required and parts of the infrastructure are automated. Review the following graph as an example of how this can reduce costs overall:



While the cost equivalent in computing power is slightly higher than using your own server, you pay for as much resource as you use, thereby reducing the average cost over a period of time. There are many different financial models for cloud vendors that offer different services.

You can use your own cloud solution, if you have the appropriate skills within your organization to ensure maximum cost-efficiency of your resource use.

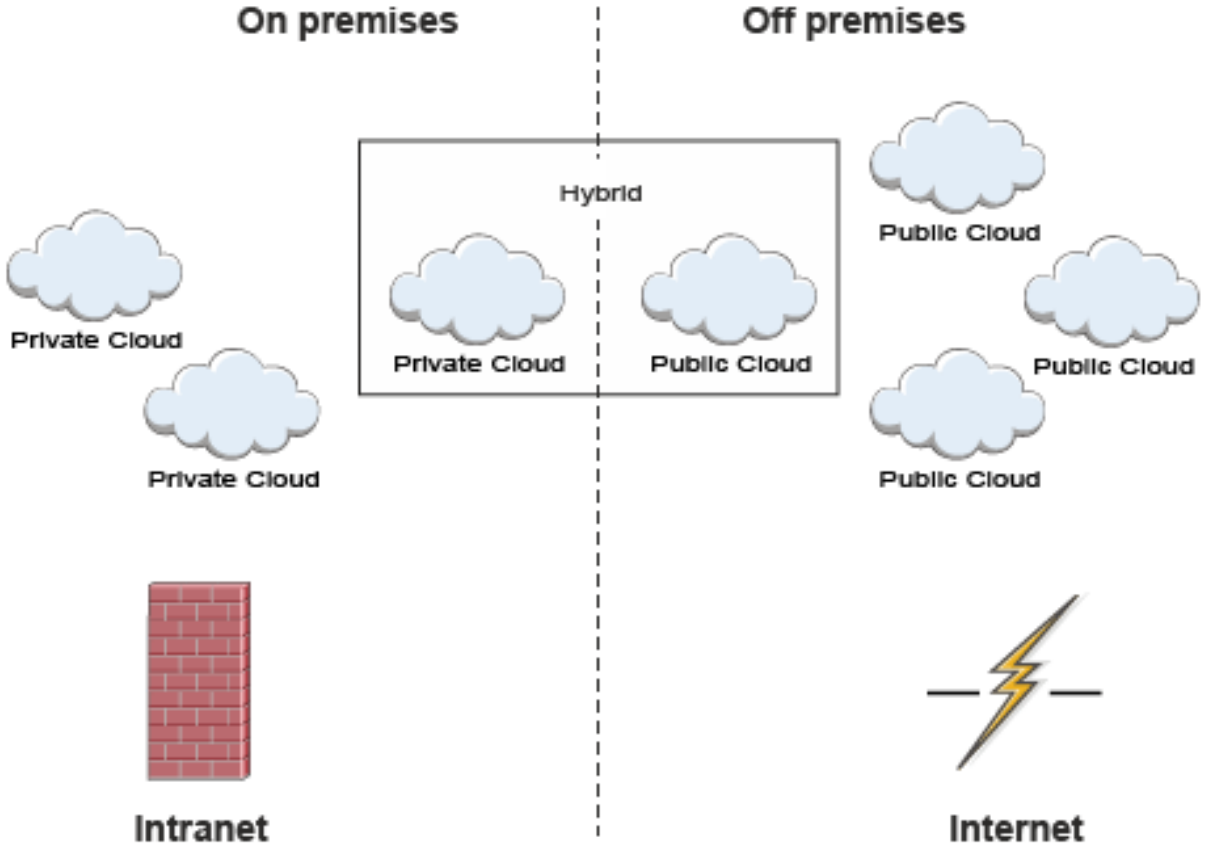
Types of cloud

You can choose how much control you retain over your cloud installation by deciding whether you want your solution to be *on premises* or *off premises*, and whether that cloud is public or private.

- An on-premises (or private) cloud is accessed and used by solely one organization. If you use a private cloud, you maintain absolute control of the maintenance and configuration, but setting up a private cloud requires significant technical skills, scrutiny of security implementation, time, and resources.

- An off premises (or public) cloud is a cloud that is available to multiple organizations, who each use parts of the cloud privately or publicly. There is little to no architectural difference to public or private clouds, except for extra and different security considerations for your services. The maintenance, configuration, and security of the hosting servers are controlled by the hosting vendor.
- A hybrid cloud service contains both on premises and off premises elements. For example, you might host client-sensitive data on premises in a private cloud, but connect to an application that is hosted in a public cloud that processes some of that data.

See the following diagram for a visual representation of the relationship between these concepts:



You can use IBM App Connect Enterprise in any of these configurations. For example, you might have a traditional installation of IBM App Connect Enterprise privately on premises, but deploy some of your integration servers in the cloud.

Cloud technologies

By using cloud technologies, you can choose to abstract parts of your network setup so that the physical components are instead virtual. You can then manipulate the non-physical components to meet the needs of your solution without the associated cost overheads of a physical change, such as changing physical servers or operating system.

This abstraction is collectively known as *XaaS* or *Everything as a Service*, and you can create any part of XaaS at different levels of system architecture.

For a cloud installation of IBM App Connect Enterprise, you can benefit from:

- Infrastructure as a Service (IaaS). IaaS is used and maintained by system architects, or the equivalent in your organization.
- Platform as a Service (PaaS). PaaS is used and maintained by application developers, or the equivalent in your organization.

Review these topics to learn about each level of XaaS and the advantages of using each layer with IBM App Connect Enterprise.

Note: Each level of XaaS is independent. You do not have to use any of these concepts together in an installation.

You can still install IBM App Connect Enterprise by using the provided installation images to a private file system. You might want to do this if you have a small-scale solution that is unlikely to change in size. To review what is required for a standard installation, see "Installing IBM App Connect Enterprise" on page 69.

Infrastructure as a Service

Learn what Infrastructure as a Service (IaaS) is, and how you might benefit from using IaaS with IBM Integration Bus.

Infrastructure as a Service acts as the equivalent to computing hardware by using virtual machines. An IaaS configuration reduces the cost overhead of managing many servers, whether on premises or off premises, as you do not need to manually configure each system. For example, you can apply security patches to all virtual machines simultaneously by using a script.

The advantage of using IaaS with IBM App Connect Enterprise is to create a scalable solution that does not require increasing configuration as it grows in size. By configuring scripts, you can scale and change attributes as required, instead of manually recoding the system infrastructure.

You can use IaaS on premises or off premises, but the primary benefit of IaaS is the ability to use a vendor to reduce the costs of maintaining and configuring the physical machines. If you want to use IaaS on premises, consider the skill availability of system architects in your organization to configure your architecture, and the associated costs.

The benefits of using an IaaS configuration are:

- Flexible (or elastic) hosting. An IaaS provider can provision new virtual machines for you quickly.
- If you use an IaaS provider, remove the responsibility and associated effort of applying fixes, security patches, and other upgrades.
- Removes or reduces capital expenditure on hardware and human resources.

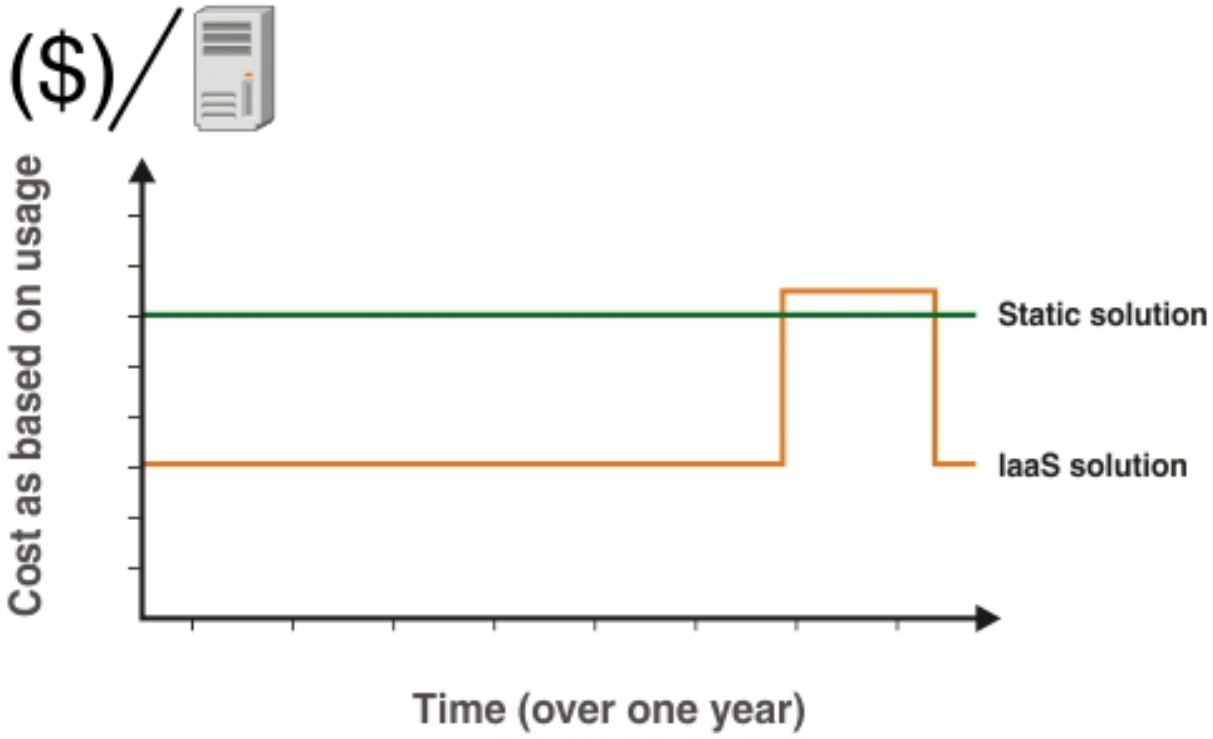
Considerations for using IaaS are:

- If you use IaaS without PaaS, the lack of dynamic scalability: you must predict your peak usage in advance.
- Choosing an IaaS vendor with the right cost model for your usage.
- The capability of the vendor to meet your ongoing business requirements.
- Security and data compliance, if you have restrictions for data use and sharing.
- Dependency on the IaaS vendor for uptime and potentially recovery.
- Requires new and different security measures.

For an example of an IaaS-based IBM App Connect Enterprise solution, review the following business context:

A retail company uses an IBM App Connect Enterprise solution that uses the IBM Retail Pack for its transaction system. The company knows that its peak message processing period is December. Their previous solution used their own on-premises servers, which met the peak demand. However, during the rest of the year, the full capacity of the servers was not used, but the company still had to pay the associating operating and maintenance costs for the entire year.

The retail company changes its infrastructure to an IaaS solution. The retail company now pays an IaaS provider for public cloud servers. The retail company pays the IaaS vendor monthly, by capacity required. The company pays for more IaaS capacity for December only to meet its peak demands. As the company now use less resources across most of the year, this solution is a cheaper option.



The graph shows how a static solution remains the same cost throughout one year, but the IaaS solution cost is consistently lower, except during peak usage. The result is that the retail company has capacity that it requires during its peak period, but does not continue to pay for that extra capacity for the rest of the year.

Chef

Chef software is an open source configuration management tool that you can use to create parts of an Infrastructure as a Service (IaaS). You can use Chef scripts to provision an IBM App Connect Enterprise installation.

Chef scripts, which are called *recipes*, are made of reusable definitions that are written in the Ruby programming language. Chef recipes that perform related functions are grouped in a single container, called a cookbook. Cookbooks and recipes automate common infrastructure tasks. The recipe definitions describe what your infrastructure consists of, and how each part of your infrastructure is deployed, configured, and managed. Chef applies these definitions to servers, to produce an automated infrastructure.

By using Chef, you can create scalable infrastructure with minimal configuration to maintain. Chef scripts form part of a recipe for your infrastructure configuration that is understood by multiple systems, services, and languages. By using Chef to configure your infrastructure deployment, you do not have to rewrite code if you change part of the underlying infrastructure, such as changing data service provider.

To learn more about Chef concepts, see the online Chef documentation About Cookbooks - Chef Docs.

Platform as a Service

Learn what Platform as a Service (PaaS) is, and how you might benefit from using PaaS with IBM App Connect Enterprise.

A PaaS cloud configuration automates the execution environment and solution deployment. For example, acting as an operating system that supports IBM App Connect Enterprise. A typical PaaS provider consists of many servers that each provide customized environments, and can dynamically allocate resources to that environment where and when they are required.

Applications are deployed into this environment, and are often charged by usage or capacity, which means that you pay only for as much as you use or specify, which can greatly reduce operating costs.

The benefits of using a PaaS configuration are:

- Reconfiguring and maintenance of the environment is faster and requires significantly less effort.
- If you use a PaaS provider, maintenance and optimization are provided, requiring less developer time.
- Dynamically scalable. PaaS automatically increases or decreases resources, which are based on demand. You usually pay only for the resources that are used.

Considerations for using PaaS are:

- Choosing a PaaS vendor with the right cost model for your usage.
- Whether the PaaS provider has any restrictions on what type of content and applications are allowed.
- What frameworks, languages and databases are supported.
- Vendor lock-in, and choosing standard or proprietary technologies. If your provider uses proprietary technologies, you might find it harder to switch vendors.
- Security and data compliance, if you have restrictions for data use and sharing.
- Dependency on the PaaS vendor for aspects of security and uptime.
- Requires new and different security measures.

Performance planning

When you design your IBM App Connect Enterprise environment and the associated resources, the decisions that you make can affect the performance of your applications.

About this task

If you are planning to create and maintain a small, stand-alone system with limited requirements for availability and performance, it is possible to achieve this relatively quickly. However, if you are planning a large or complex system, or if you have specific requirements for high availability or performance, it is worth taking time to understand the factors that can influence your design and the techniques that you can use to optimize it. As a starting point, consider the following aspects of your system and understand how these factors can influence performance:

Message flows

A message flow includes an input node that receives a message from an application over a particular protocol; for example, IBM MQ. The message must be parsed by the input node, and the performance impact of this parsing varies according to the parser that is used and the number of parses required. You can reduce the impact of parsing by using some optimization techniques such as parsing avoidance or partial parsing. For more information about parsing, see “Parsing and message flow performance” on page 2697.

The number and length of message flows have implications for performance, and it is important to keep the number of nodes to a minimum. See “Message flow design and performance” on page 2689 for more information about optimizing message flow performance. Other aspects of processing in a message flow that might affect performance are the amount, efficiency, and complexity of ESQL, access to databases, and how many message tree copies are made. For more guidance about these factors, see “Code design and performance” on page 2690.

It is also important to consider how you split your business logic; how much work should the application do, and how much should the message flow do? Every interaction between an application and a message flow involves I/O and message parsing, and therefore adds to processing time. Design your message flows, and design or restructure you applications, to minimize these interactions.

Messages and message models

The type, format, and size of the messages that are processed can have a significant effect on the performance of a message flow. For example, if you process persistent messages, they must be stored for safekeeping.

If you do not plan to interrogate the structure, you can use the BLOB domain.

If you are working with general text or binary messages, then you need to create a DFDL (or MRM) model to describe the message. The organization of the model can have a significant effect on performance. For fields that are choices, are optional, or are in arrays, the model must identify the field as efficiently as possible. Use tags (named initiators in DFDL) if they are available. To resolve a choice, DFDL provides a direct dispatch mechanism, which avoids parsing each branch in turn if there is a field that indicates which branch to take. If you are using a DFDL discriminator, ensure that the discriminator is placed so that it is evaluated as early as possible. While regular expressions can be used to identify and extract fields, they are generally slower than other techniques.

If you are working in XML, be aware that it can be verbose, and therefore produce large messages. However, XML message content is easier to understand than other formats, such as binary fixed-length format.

For more information about these factors, see Performance considerations for regular expressions in TDS messages.

Integration node configuration

You can create and configure one or more integration nodes, on one or more computers, and for each integration node you can create multiple integration servers, and multiple message flows. Your configuration decisions can influence message flow performance, and how efficiently messages can be processed.

For more information about these factors, see “Tuning the integration node” on page 2762, and “Optimizing message flow throughput” on page 2763.

Planning for security

It is important to be aware of security issues when you are planning to install and use IBM App Connect Enterprise.

About this task

On Linux® and UNIX systems, you must complete security tasks before you install IBM App Connect Enterprise. On Windows systems, security tasks are completed during and after installation.

After installation, refer to one of the following documents for further security considerations:

- For Windows, Linux and UNIX systems, see “Creating user IDs” on page 2633.

For an introduction to various aspects of security, see “Security overview” on page 2491.

Replicating your environment for multiple users

You can ensure that your environment is more easily reproducible by using scripting and virtualization, in addition to documentation.

About this task

You can create environment instances by using Chef scripts to automate installation. For more information about Chef, see “Chef” on page 15.

Procedure

1. Download the IBM App Connect Enterprise installation images.
2. Download the Chef Cookbook for IBM App Connect Enterprise from GitHub.
3. Add the IBM App Connect Enterprise cookbook to the Chef server.
   Alternatively, if you are using Chef solo, add the cookbook to your file system.
4. Set the recipe attributes to point to the FTP or HTTP server.

5. Run the Chef scripts.

Chapter 2. Migrating

To migrate an integration server or integration node from IBM App Connect Enterprise 11.0 or IBM Integration Bus 10.0 to IBM App Connect Enterprise 12.0, plan your migration strategy, perform pre-migration tasks, migrate your domain components, or server configuration and resources, then complete post-migration tasks.

Before you begin

Tip: Check for migration information updates on the IBM App Connect Enterprise support web page.

About this task

Video: IBM App Connect Enterprise - Migration and Extraction

In this demo, Ben Thompson gives an overview of migrating from IBM Integration Bus V10 to IBM App Connect Enterprise V11.

Note:

This video was created for App Connect Enterprise 11.0, but also applies to App Connect Enterprise 12.0.

Preparing for migration

Plan the order and extent of the migration of components and resources to IBM App Connect Enterprise 12.0.

About this task

To prepare for migration, complete the following steps.

Procedure

1. Check that your current installation of IBM App Connect Enterprise Version 11.0 or IBM Integration Bus 10.0 is at a supported level for migration (see “Supported migration paths” on page 20).

   For the latest details of all supported levels of hardware and software, see the IBM App Connect Enterprise system requirements website.
2. Some function that was available in IBM Integration Bus is not currently available in IBM App Connect Enterprise 12.0; elements of this function are to be made available in fix pack updates. For details of which commands, policy types, and other resources are unavailable in IBM App Connect Enterprise at this fix pack release, see the Frequently asked questions about website.
3. Determine your migration priorities by following the steps in “Migration planning” on page 20.
4. Review the options that are available to install IBM App Connect Enterprise 12.0 components on the same computer as components from IBM App Connect Enterprise Version 11.0 or IBM Integration Bus 10.0 (see “Coexistence of IBM App Connect Enterprise 12.0 with previous versions” on page 70).
5. Learn about new and changed function in Version 12.0 by reading What's new in ?.

   These changes might affect how you want to use your migrated components in the future.
6. Review the IBM MQ configuration options that you can include in your migration plan in “IBM MQ and migration to IBM App Connect Enterprise 12.0” on page 28.

   These changes might affect your choice of integration node migration option.
7. Review the options for migrating your integration nodes and integration servers, and decide whether to use extract migration or parallel migration (see “IBM App Connect Enterprise migration options” on page 29).

8. Review the technical changes in behavior in Version 12.0 in "Behavioral changes in Version 12.0" on page 22.

   These changes might affect your post-migration development tasks.
9. Check the requirements for other products on which Version 12.0 components might depend.

   If you configured your message flows to use external resources, such as databases or event monitoring applications, you might have to modify your configuration. You can find details of supported versions of complementary products on the IBM App Connect Enterprise system requirements web page.

What to do next

After you plan your migration, complete the pre-migration tasks by following the instructions in "Performing pre-migration tasks" on page 41.

Supported migration paths

You can migrate to IBM App Connect Enterprise 12.0 from IBM App Connect Enterprise Version 11.0 and IBM Integration Bus 10.0.

Note: For the latest details of all supported levels of hardware and software, see the IBM App Connect Enterprise system requirements website.

You can migrate from IBM App Connect Enterprise 11.0 or IBM Integration Bus 10.0 to either the Advanced Edition or Standard Edition of IBM App Connect Enterprise 12.0; do not migrate integration nodes or servers to the Developer Edition.

You can use either parallel or extract migration to migrate integration nodes and managed integration servers to IBM App Connect Enterprise 12.0, as described in "Performing parallel migration for an integration node " on page 61 and "Performing extract migration of an integration node or integration server" on page 57.

Independent integration servers that were created in IBM App Connect Enterprise 11.0 can be started in IBM App Connect Enterprise 12.0 without the need to migrate them or their resources. However, integration server work directories are typically created as transient rather than long-term artifacts, and many users prefer to rebuild them when moving from one version to another.

You can use the IBM Integration Explorer view of the IBM App Connect Enterprise Toolkit to add connections for integration servers and integration nodes. IBM App Connect Enterprise Toolkit Version 12.0 supports Version 12.0 integration servers and integration nodes only. Connections to integration servers and integration nodes that are running on previous releases are not supported.

Migration planning

The order in which you migrate your IBM App Connect Enterprise 11.0 or IBM Integration Bus 10.0 environment depends on whether your priority is to include new development features, take advantage of new operational features, or simply implement a fully supported version of IBM App Connect Enterprise 12.0.

If you are migrating from IBM Integration Bus 10.0, your existing environment might consist of any of the following components:

- Integration nodes that support production applications
- Build systems that create deployable resources from the development source files
- Integration nodes that are used for testing applications
- Integration nodes that are used for developing applications
- Instances of the IBM Integration Toolkit

The order in which you migrate your environment to IBM App Connect Enterprise 12.0 is likely to depend on which of the following factors is most important to you:

- “Supported version or new operational features” on page 21
- “New application features” on page 21

In both circumstances, you can use parallel migration. If you are migrating an integration server, you can use the **mqsiextractcomponents** command to migrate the configuration and resources that exist for your integration server to IBM App Connect Enterprise 12.0.

Supported version or new operational features

If your priority in migrating to IBM App Connect Enterprise 12.0 is simply to have an environment that is at a fully-supported version of IBM App Connect Enterprise 12.0, and you do not need to use any of the new Version 12.0 features immediately, there is a minimum number of steps you must complete.

If your priority in migrating to IBM App Connect Enterprise 12.0 is to use the new Version 12.0 operational features, you can update your integration nodes first. You can use existing development environments and application build processes, and deploy your existing BAR files until you are ready to migrate your development resources.

In either scenario, you migrate the components of your environment in the following order:

1. Migrate the integration nodes that support your test environment.
2. Implement new Version 12.0 operational functionality on your test environment or, at a minimum, update existing operational functionality:
   - If you are using IBM Integration Explorer in your existing IBM Integration Bus environment, define new operational procedures that use the web user interface. You cannot use previous versions of IBM Integration Explorer to administer IBM App Connect Enterprise 12.0.
   - If you are using scripts to administer your existing environment, update any scripts that use commands that connect to integration nodes. The parameters in IBM Integration Bus 10.0 that are used by commands that connect to integration nodes have changed in Version 12.0.
3. Migrate the integration nodes that support your production environment.
4. Implement new Version 12.0 operational functionality on your production environment or, at a minimum, update existing operational functionality.
5. If you have any integration nodes that support your development environment, migrate these integration nodes to Version 12.0.
6. Update your build system to create Version 12.0 deployable resources. If required, update build scripts to take advantage of new Version 12.0 operational functionality but, at a minimum, update any scripts that use commands that connect to integration nodes.
7. Install IBM App Connect Enterprise 12.0 on your developer workstations. If you cannot migrate all developer workstations at the same time, you must create separate development streams. You cannot use Version 12.0 development tools to build applications for an environment that is running a previous version of IBM App Connect Enterprise or IBM Integration Bus.
8. Import development resources from your previous Toolkit.

To migrate integration nodes, use parallel migration (see “IBM App Connect Enterprise migration options” on page 29).

When you have imported all the development resources, you can uninstall the previous versions of IBM Integration Toolkit, and any integration nodes that you do not want to migrate.

New application features

If your priority in migrating to IBM App Connect Enterprise 12.0 is to develop applications that take advantage of new functions and features in Version 12.0, you can install a new development environment alongside your existing development environment, and create new build, test, and production environments to support your Version 12.0 development.

In this scenario, you migrate the components of your environment in the following order:

1. Install IBM App Connect Enterprise 12.0 on your developer workstations. To maintain existing applications in your existing environment while you are building new applications for Version 12.0, you must run two development streams. You cannot use Version 12.0 development tools to build applications for an environment that is running a previous version of IBM App Connect Enterprise or IBM Integration Bus.
2. If you are updating an existing application, import development resources from your previous Toolkit.
3. Develop applications that take advantage of the Version 12.0 features.
4. Create a new build system that creates Version 12.0 deployable resources. You can use build scripts from your previous version but you must update any scripts that use commands that connect to integration nodes. The parameters in IBM Integration Bus 10.0 that are used by commands that connect to integration nodes have changed in Version 12.0.
5. Create one or more IBM App Connect Enterprise 12.0 integration nodes to support the testing of the Version 12.0 applications.
6. Update existing operational functionality:
   - If you are using IBM Integration Explorer in your existing IBM Integration Bus environment, define new operational procedures that use the web user interface. You cannot use previous versions of IBM Integration Explorer to administer IBM App Connect Enterprise 12.0.
   - If you are using scripts to administer your existing environment, update any scripts that use commands that connect to integration nodes.
7. Deploy Version 12.0 applications to the Version 12.0 testing environment as required.
8. Create one or more IBM App Connect Enterprise 12.0 integration nodes to support production use of the Version 12.0 applications.
9. Deploy Version 12.0 applications to the Version 12.0 production environment as required.
10. Migrate or deprecate applications from the original environment as required.

This type of migration is known as parallel migration (see “IBM App Connect Enterprise migration options” on page 29).

When all applications are migrated to Version 12.0, you can uninstall the original environment.

Behavioral changes in Version 12.0

Depending on your current version of IBM Integration Bus or App Connect Enterprise, IBM App Connect Enterprise 12.0 might introduce technical changes in behavior. These changes might affect your post-migration development tasks.

Review the following changes to see how your post-migration development tasks might be affected:

Platform support

IBM App Connect Enterprise 12.0 is available on:

- Windows
- Linux
- AIX®

For more information about the operating systems that are supported, see IBM App Connect Enterprise 12.0 system requirements.

Policies control and update message flow and message flow node properties

In IBM Integration Bus 10.0, configurable services were used to control and update connection properties and other operational properties of message flows and message flow nodes at run time. In IBM App Connect Enterprise 12.0, you can use policies for these tasks.

You can migrate your existing configurable policies by using the **mqsiextractcomponents** command. If you are performing parallel migration, you need to create policies to replace your configurable services.

For more information, see "Overriding properties at run time with policies" on page 324.

Nodes that are no longer available

Your message flows from earlier versions work in IBM App Connect Enterprise 12.0. However, a number of message flow nodes are not available in Version 12.0; if your earlier message flows include them, consider reworking them.

The following nodes are not available in IBM App Connect Enterprise 12.0:

- DataDelete node
- DataInsert node
- DataUpdate node
- Extract node.
- Mapping node from WebSphere® Message Broker Version 7.0 and earlier versions
- MQOptimizedFlow node
- Real-timeOptimizedFlow node
- Warehouse node

If you try to use a message flow that contains any of these nodes, the message flow does not start and message BIP2355 is written to the syslog. The message mapping (`.msgmap`) on these nodes can be viewed but they cannot be deployed to the run time environment in IBM App Connect Enterprise 12.0.

For more information, see Built-in nodes.

IBM App Connect Enterprise 12.0 REST administration API (REST API) Version 2 is available to issue administration commands

IBM App Connect Enterprise 12.0 does not include the IBM Integration API. Instead, you can use the REST API (Version 2) that is provided in IBM App Connect Enterprise 12.0 to issue administration commands.

For more information, see "Managing resources by using the administration REST API" on page 334.

Web user interface extended to support administration tasks

IBM App Connect Enterprise 12.0 does not include IBM Integration Explorer. Instead, the web user interface is extended to support integration node administration tasks such as creating and managing integration servers, and deploying and managing resources.

For more information, see web user interface.

Keystore format for administration security

The certificate store for use in securing the REST Administration port, which is used by the Web User Interface and IBM App Connect Enterprise Toolkit must be a `.pem` `.p12` or `.pfx` format. This certificate store cannot be in JKS format, and hence must be a separate certificate store from the certificate store that used by the integration server for things such as HTTPS message flow nodes.

For more information, see "Authorizing users for administration" on page 2516.

Additional options for administration security

In IBM App Connect Enterprise 12.0, you have two options for configuring administration security. As in previous versions, you can configure queue-based permissions by using IBM MQ queues on the queue manager that is specified on the integration node. Alternatively, you can configure file-based permissions on your integration node, which you set by using the **mqsichangefileauth** command.

If you configure administration security, then you perform parallel migration to integration nodes that are not configured with queue managers, you must reconfigure your administration security to use file-based permissions. Any security permissions that are set before migration are not retained after migration.

For more information, see "Authorizing users for administration" on page 2516.

Administration scripts might need to be updated

A number of commands are not available in this release. In addition, some parameters for administration commands that connect to integration nodes and integration servers were changed. For more information, see commands. The **MQBrokerConnectionParameters** IBM Integration API class is also deprecated (see the Javadoc for the IBM Integration API).

IBM App Connect Enterprise Toolkit help topics might include details of function that is not available to you

By default, the IBM App Connect Enterprise Toolkit is now configured to use online product documentation to provide context-sensitive help from the latest information that is available. In previous versions, the Toolkit contained local help documentation, which was not automatically updated when new information became available. The way that help is displayed in the IBM App Connect Enterprise Toolkit is not changed. However, the content of the IBM App Connect Enterprise section of the online product documentation is updated when fix packs are made available. If you do not deploy the latest fix pack, the help information might include details of function that is not available to you.

If you do not want to access the online product documentation, you can download and install a local source of product documentation. For more information about local documentation options, see Adding documentation to the IBM App Connect Enterprise Toolkit.

Message maps: change in behavior

In IBM App Connect Enterprise 12.0, the behavior of the **Assign** transform in the Graphical Data Mapping editor has been corrected when you assign an empty string to an element.

- In previous versions of IBM Integration Bus, when you perform an **Assign** transform on a target element that is defined as **xsd:string**, the Graphical Data Mapping editor sets the internal `NULL` value in the message tree for that element.
- In IBM App Connect Enterprise 12.0, when you perform an **Assign** transform on a target element that is defined as **xsd:string**, the Graphical Data Mapping editor sets the element value to the empty string `''` value.

For message formats such as XML, this change in behavior has no impact on the message flow. However, if your message flow has logic that tests that the value of an element is the internal `NULL` value, you must

modify the test to look for the empty string value. Alternatively, you must modify the map to use the new **iib:nullvalue()** function. To call the new function, you use a **Custom XPath** transform.

Message maps are validated at deployment time

In IBM App Connect Enterprise 12.0, message maps are prepared for execution on deployment instead of when the first message flows through the Mapping node. For more information, see Deploying message maps.

Java isolation is active and cannot be disabled

Previous versions of IBM Integration Bus did not support Java isolation in applications. All Java that was deployed to the execution group was loaded into a single class loader, which was used by all JavaCompute nodes in all applications. This behavior precluded the use of duplicate classes.

When you create an application in IBM Integration Bus 10.0, Java isolation is enabled by default. For each application, a Java class loader is built that contains only the Java that is deployed in that application and any included static libraries.

When you create an application in IBM App Connect Enterprise 11.0 or later, Java isolation is active and you cannot disable it.

Publication of statistics to the web user interface is enabled by default

When an integration node or integration server is created in IBM App Connect Enterprise Version 11.0.0.8 or later, the publication of resource statistics and message flow snapshot statistics is activated by default. The publication of archive statistics is turned off by default. The **publicationOn** property in the `server.conf.yaml` or `node.conf.yaml` file is explicitly set to `active` and the **outputFormat** property is set to `json`, which enables the publication of snapshot data to the web user interface. The **reportingOn** property is explicitly set to `true`, which enables resource statistics for the integration node or server to be published automatically to the web user interface.

For integration nodes and servers that were created before IBM App Connect Enterprise Version 11.0.0.8, the publication of all statistics was turned off by default. To enable the publication of statistics for integration nodes or servers that were created before V11.0.0.8, edit the relevant `.conf.yaml` file and activate the **publicationOn** and **reportingOn** properties, as required.

For more information, see “Managing resource statistics collection” on page 2741 and “Configuring the collection of message flow statistics by using a .yaml configuration file” on page 2729.

UUIDs are not assigned to the integration node, integration server, application, or message flow components

In IBM App Connect Enterprise 11.0 and later, UUIDs are not assigned to the integration node, integration server, application, or message flow components. When these fields are displayed (in monitoring events, for example), they are set to all zeros.

For more information, see “The monitoring event” on page 2778.

Message flow monitoring

Monitoring events are issued in a new format that provides additional information. However, the format that was used in IBM Integration Bus 10.0 can also be used in IBM App Connect Enterprise 12.0 by editing the `eventFormat` property in the `Monitoring.MessageFlow` section of the `.conf.yaml` file.

For more information, see “Activating monitoring” on page 2804.

LDAP authentication and local passwords

If a web user account has a local password, and LDAP authentication is enabled, the local password is ignored. When LDAP authentication is enabled, all web user logins must be authenticated by using LDAP.

Any local passwords are ignored. For more information, see “Enabling LDAP authentication” on page 2511 and command - , , and systems.

HTTPConnector and HTTPSConnector policies

HTTPConnector and HTTPSConnector policies were deprecated in IBM App Connect Enterprise 11.0 Fix Pack 5 and are not supported in IBM App Connect Enterprise 12.0. If you try to deploy one of these policy types in IBM App Connect Enterprise 12.0, you see an error and the policies are not loaded or used. Instead, configure the HTTPConnector or HTTPSConnector Resource Manager section of the `server.conf.yaml` file.

Enhanced flexibility in interactions with IBM MQ

Greater flexibility was introduced in IBM Integration Bus 10.0 in its interactions with IBM MQ. IBM App Connect Enterprise 12.0 maintains this enhanced flexibility.

You can configure local or client connections to IBM MQ, enabling your integration nodes to get messages from, or put messages to, queues on a local or remote queue manager. You can configure either a local or client connection between your integration node and your queue manager, depending on the configuration of your existing architecture. If your IBM MQ queue manager is running on the same machine as your integration node, you can specify a local connection to the queue manager. Alternatively, if the IBM MQ queue manager that you want to connect to is hosted on a separate machine from IBM App Connect Enterprise, you can configure a client connection from your integration node so that it can access the messages on the remote queue manager.

When you configure a connection from an MQ node to an IBM MQ queue manager, you can optionally configure the connection to use a security identity for authentication, SSL for confidentiality, or both. The security identity, which passes user name and password security credentials to the queue manager, can be used on connections to local or remote queue managers. For connections to remote queue managers, you can choose whether to use the SSL protocol to provide confidentiality on the client connection. IBM App Connect Enterprise supports a subset of the SSL functionality that is supported by IBM MQ. For more information, see “Connecting to a secured IBM MQ queue manager” on page 747.

Note: You cannot use a secured queue manager as the local default queue manager for an integration node or an integration server.

You can get messages from, or put messages to, IBM MQ queues on local or remote queue managers, by configuring the connection properties of the following MQ nodes:

- node
- node
- node
- node

Alternatively, you can specify a queue manager to be associated with the integration node by using the **-q** parameter on the **mqsicreatebroker** command. This queue manager is used by default for MQ processing in the message flow if no queue manager is specified explicitly on the MQ node. This queue manager is also used by some message flow nodes that require a queue manager to be specified on the integration node, such as the event-driven processing nodes used for aggregation, timeout, message collection, and message sequencing. The **-q** parameter specifies the name of a queue manager, but it does not create the queue manager automatically. You must define and start the queue manager as separate tasks, in addition to specifying it with the **mqsicreatebroker** command.

Message flows can contain multiple MQInput and MQOutput nodes, each of which can access different queue managers specified in the MQ node. For more information, see node and node.

IBM MQ is not a prerequisite for using IBM App Connect Enterprise, which means that you can develop and deploy applications with IBM App Connect Enterprise independently of IBM MQ. However, some IBM App Connect Enterprise features require access to IBM MQ, including the MQ nodes and the event-driven processing nodes that are used for aggregation and timeout flows, message collections, and message sequences. IBM MQ is not provided as part of the IBM App Connect Enterprise installation package;

however, when you purchase a license for IBM App Connect Enterprise, your license entitles you to install IBM MQ for use by App Connect Enterprise, within the terms of the license.

The following IBM App Connect Enterprise features require IBM MQ Server to be installed on the same machine as the integration node, and they are available for use only if you specify a queue manager on the integration node:

- Queue-based administration security (MQ is not required for file-based security)
- Global transactionality
- FTEInput and FTEOutput nodes
- CDInput and CDOutput nodes
- Integration nodes with HTTP listeners
- HTTP proxy servlet
- High availability configurations

The integration node listener requires access to IBM MQ Server, so you must install it if you want to use an integration node listener to manage HTTP messages in your HTTP or SOAP flows. However, if you use HTTP nodes or SOAP nodes with the integration server embedded listener, they do not require access to IBM MQ.

The following IBM App Connect Enterprise features require access to system queues on a local queue manager (running on the same machine as the integration node) for the storage and retrieval of state information:

- Queue-based administration security (system queues are not required for *file-based* security).
- Integration node HTTP listener.
- HTTP proxy servlet.

The following IBM App Connect Enterprise features require access to system queues on a either a local or remote queue manager for the storage and retrieval of state information:

- Record and replay.
- Event-driven processing nodes (aggregate, collector, sequence, resequence, and timeout nodes).

For information about creating the system queues, see “Creating the default system queues on an IBM MQ queue manager” on page 99.

On Linux and AIX systems only, you must also configure the IBM MQ environment that you want the integration node to use before you start it. If you do not set the environment, your integration node might not run in the expected location. For more information, see “Setting the IBM MQ environment on Linux and AIX” on page 84.

The MQInput, MQOutput, MQGet, and MQReply nodes require that IBM MQ is installed either locally or remotely, but they do not require a queue manager to be specified on the integration node unless you want to use this queue manager by default for your local MQ connection. For more information, see “Configuring a local connection to IBM MQ” on page 744 and “Configuring a client connection to IBM MQ” on page 745.

The SAPInput, SAPReply, and SAPRequest nodes require that either IBM MQ Client or Server is installed on the same machine as the integration node, and they require a queue manager to be specified on the integration node.

For a list of the main IBM App Connect Enterprise features, including information about the features that require the installation of IBM MQ Client or Server, see features.

For a summary of the features that are new to IBM App Connect Enterprise 12.0, see What's new in ?.

IBM MQ and migration to IBM App Connect Enterprise 12.0

During migration, you might want to configure IBM App Connect Enterprise 12.0 to take advantage of the increased flexibility that was introduced in IBM Integration Bus 10.0 for its interactions with IBM MQ.

Greater flexibility was introduced in IBM Integration Bus 10.0 in its interactions with IBM MQ; IBM App Connect Enterprise 12.0 maintains this enhanced flexibility. See “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “IBM MQ topologies” on page 739.

In IBM Integration Bus 10.0, the integration layer of your architecture must contain IBM MQ (or IBM MQ) queue managers. If you have queues that you use in integration applications, you must have an existing IBM MQ (or IBM MQ) topology in which application messages are routed to the queue manager that is specified on the integration node by using IBM MQ (or IBM MQ) channels, remote queue definitions, and distributed messaging. It might be possible to simplify your system so that your message flows interact directly with remote queue managers, which might simplify the topology that you need to manage. This simplification requires that you redesign your message flows and your topology, and is more than just a migration of your existing solution. However, you might want to include these activities as part of your migration plans.

To determine which components of your existing deployment are using capabilities that require IBM MQ or (IBM MQ) queue managers to be a part of the integration layer of your IBM App Connect Enterprise 12.0 architecture, and which components are using capabilities that can integrate with IBM MQ application queues on a remote queue manager, see “Installing IBM MQ” on page 96.

Depending on whether you can change your IBM MQ topology during the migration process, you have the following options for migrating integration nodes:

- Create a new Version 12.0 integration node and associate the integration node with a new queue manager. Add the new queue manager into your IBM MQ network, migrate the applications from the original integration node, and redirect application queues as required (parallel migration).
- Create a new Version 12.0 integration node and associate the integration node with the same queue manager as the original integration node. Migrate the applications from the original integration node, and keep the IBM MQ application endpoints and IBM MQ topology the same (parallel migration).

  Note: This option cannot be used if your integration node has message flows that contain the following message flow nodes:

  – AggregateControl
  – AggregateReply
  – AggregateRequest
  – Collector
  – Sequence
  – Resequence
  – TimeoutControl
  – TimeoutNotification

  These nodes use MQ queues to save state, which cannot be shared between integration nodes.
- Create a new Version 12.0 integration node and do not associate the integration node with a queue manager. Migrate the applications from the original integration node, and configure message flows that require IBM MQ queues to connect to specified queue managers, either by directly configuring the message flow or by using a policy (parallel migration).

For more information, see “IBM App Connect Enterprise migration options” on page 29.

IBM App Connect Enterprise migration options

IBM App Connect Enterprise 12.0 supports parallel migration and extract migration to migrate your application logic to Version 12.0 systems.

If you want to reproduce the integration node function on another computer, you can use *parallel migration* to associate the application logic on your existing integration node with a separate Version 12.0 integration node. For more information, see “Performing parallel migration for an integration node ” on page 61.

By using *extract migration*, you can migrate the configuration and resources of existing integration nodes and servers to IBM App Connect Enterprise 12.0, as described in “Performing extract migration of an integration node or integration server” on page 57.

Independent integration servers that were created in IBM App Connect Enterprise 11.0 can be started in IBM App Connect Enterprise 12.0, without the need to migrate them or their resources. However, integration server work directories are typically created as transient rather than long-term artifacts, and many users prefer to rebuild them when moving from one version to another.

Restrictions

Consider the following restrictions before you decide whether to use parallel migration for your integration nodes.

IBM MQ enhancements

Greater flexibility was introduced in IBM Integration Bus 10.0 in its interactions with IBM MQ; IBM App Connect Enterprise 12.0 maintains this enhanced flexibility. Ensure that you are using a supported version of IBM MQ to take advantage of the greater flexibility.

If you use parallel migration, you can install and configure an IBM App Connect Enterprise 12.0 deployment that takes advantage of the IBM MQ flexibility, and then move your applications into the Version 12.0 deployment.

For more information, see “IBM MQ and migration to IBM App Connect Enterprise 12.0” on page 28.

Maintaining administration security

If you have administration security configured in your current environment by using IBM MQ queues, and you perform parallel migration, you must re-create the equivalent administration security settings by using file-based permissions on your new integration nodes (see “Authorizing users for administration” on page 2516).

Example migration scenarios

Example migration scenarios demonstrate how to migrate message flows and applications from IBM Integration Bus 10.0 to IBM App Connect Enterprise 12.0.

- “Migrating an integration node with configurable services ” on page 30

  This scenario shows how to migrate an IBM Integration Bus 10.0 integration node that has configurable services for external resources. The Version 10.0 node also has a specific user ID and password that is associated with one or more resources by using the **mqsisetdbparms** command.

- “Migrating an integration node with multiple integration servers” on page 31

  This scenario uses the **mqsiextractcomponents** command to migrate an IBM Integration Bus 10.0 integration node with multiple integration servers to IBM App Connect Enterprise 12.0 independent integration servers.

- “Migrating an integration server that contains applications and a shared library” on page 33

  This scenario shows how to migrate an IBM Integration Bus 10.0 integration server with applications and a shared library to IBM App Connect Enterprise 12.0.

- "Migrating an SAP integration flow" on page 34

  This scenario shows how to migrate an SAP integration flow that runs on IBM Integration Bus 10.0.

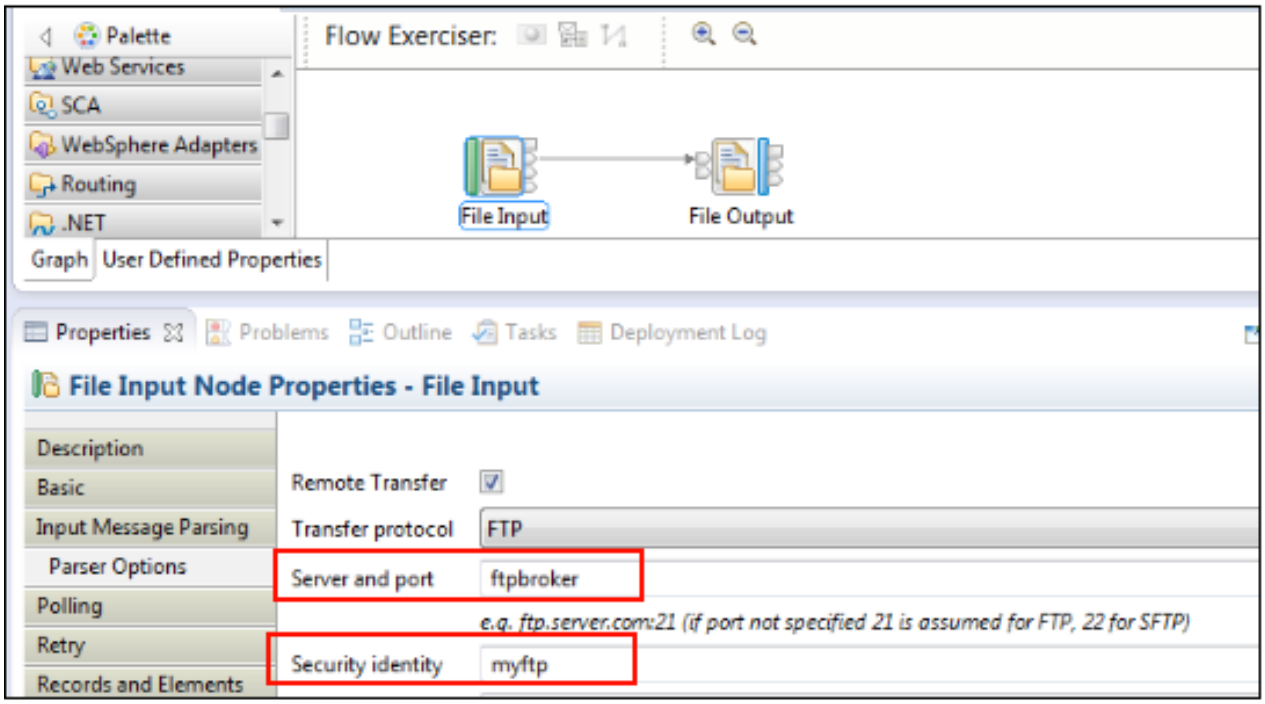
Migrating an integration node with configurable services

Migrate an IBM Integration Bus 10.0 integration node that has configurable services for external resources, and a specific user ID and password that is associated with one or more resources by using the **mqsisetdbparms** command.

About this task

In this scenario, a simple flow that contains a FileInput node and a FileOutput node is deployed to an IBM Integration Bus 10.0 integration server, then to an integration node. An FtpServer configurable server has been created by using the following command:

```
mqsicreateconfigurableservice MigrationNode -c FtpServer -o FtpServer01
-n serverName,scanDelay,transferMode,connectionType,securityIdentity
-v one.hursley.abc.com:123,20,Binary,ACTIVE,myftp
```



The following steps describe how to migrate the flow to IBM App Connect Enterprise 12.0.

Procedure

1. Back up the IBM Integration Bus 10.0 integration node by using the **mqsibackupbroker** command.

   ```
   mqsibackupbroker IntegrationNode -d  directory
   ```

   A file with a name in the format `intNodeName_yyMMdd_HHmmss.zip` is created in the specified directory.
2. In IBM App Connect Enterprise 12.0, use the **mqsicreatebroker** command to create a new integration node.

   `mqsicreatebroker IntegrationNode`
3. Start the integration node by running the command `mqsistart IntegrationNode`.

4. Create the directory structure `servers\IntegrationServer` under the integration node work path ($MQSI_WORKPATH).
   For example, on Windows: `C:\ProgramData\IBM\MQSI\components\IntegrationNode\servers\IntegrationServer`.
5. Stop the integration node by running the command `mqsistop IntegrationNode`.
6. Use the **mqsiextractcomponents** command to migrate the IBM Integration Bus 10.0 integration server and resources to IBM App Connect Enterprise 12.0.

   All deployed resources are migrated under the `/run` directory: `C:\ProgramData\IBM\MQSI\components\IntegrationNode\servers\IntegrationServer\run`.

   During migration, the IBM Integration Bus 10.0 configurable services are converted to policies in IBM App Connect Enterprise 12.0. The policy project resides in the `/run` subfolder in the work directory. No value was specified for the `policy_project_name`; therefore, policies are created in a policy project called `DefaultPolicies`. For example, if the FTPServer configurable service name is `ftpbroker`, a policy file with name `ftpbroker.policyxml` is created under the `run\DefaultPolicies` folder in the work path.
7. Start the integration node by running the command `mqsistart V12BRK`.

   You might see the following error message in the event viewer:

   ```
   File node "File Output" in message flow "FileInOutFlow". The remote user identifier supplied
    as "myftp" is invalid.
   The user identifier supplied by a securityIdentity is not valid. Either the user identifier
    is missing, or no securityIdentity definition exists, or the securityIdentity registry
    information could not be read due to a permissions problem. FTP processing for this node has
    been disabled.
   Ensure that the securityIdentity is correctly defined using the mqsisetdbparms command.
   ```

   Currently, the **mqsiextractcomponents** command does not migrate the credentials that are set by using the **mqsisetdbparms** command in IBM Integration Bus 10.0. Therefore, you need to set the security identity and other credentials by running the **mqsisetdbparms** again in IBM App Connect Enterprise 12.0:

   ```
   mqsisetdbparms V12BRK -n ftp::myftp -u user -p password
   ```
8. Restart the IBM App Connect Enterprise 12.0 integration node by running the **mqsistop** command, followed by the **mqsistart** command.

Migrating an integration node with multiple integration servers

Use the **mqsiextractcomponents** command to migrate an IBM Integration Bus 10.0 integration node with multiple integration servers to IBM App Connect Enterprise 12.0 independent integration servers.

Procedure

1. Create an application with a few message flows in the IBM Integration Bus 10.0 Toolkit.
2. Create an integration node called `Migration Node`.
3. Create three integration servers called `MigrationServer1`, `Migrationserver2`, and `MigrationServer3`.
4. Deploy the application to all of the three integration servers.
5. Back up the integration node by using the **mqsibackupbroker** command:

   ```
   mqsibackupbroker MigrationNode -d C:\temp\
   ```
6. Copy the broker backup file to the IBM App Connect Enterprise 12.0 server for migration.
7. In IBM App Connect Enterprise 12.0, use the **mqsiextractcomponents** command to create three independent integration servers:

   ```
   mqsiextractcomponents --source-integration-node MigrationNode --source-integration-server
    MigrationServer1 --backup-file C:\temp\MigrationNode_181114_120112.zip --target-work-
   directory C:\temp\MigrationServer1 --default-application DefApp
   ```

```
mqsiextractcomponents --source-integration-node MigrationNode --source-integration-server
 MigrationServer2 --backup-file C:\temp\MigrationNode_181114_120112.zip --target-work-
directory C:\temp\MigrationServer2 --default-application DefApp
mqsiextractcomponents --source-integration-node MigrationNode --source-integration-server
 MigrationServer3 --backup-file C:\temp\MigrationNode_181114_120112.zip --target-work-
directory C:\temp\MigrationServer3 --default-application DefApp
```

Note: We specify the –`default-application DefApp` option to ensure that any message flows that are deployed as independent projects are moved into the default application `DefApp`. In IBM App Connect Enterprise 12.0, all resources must be part of an application.

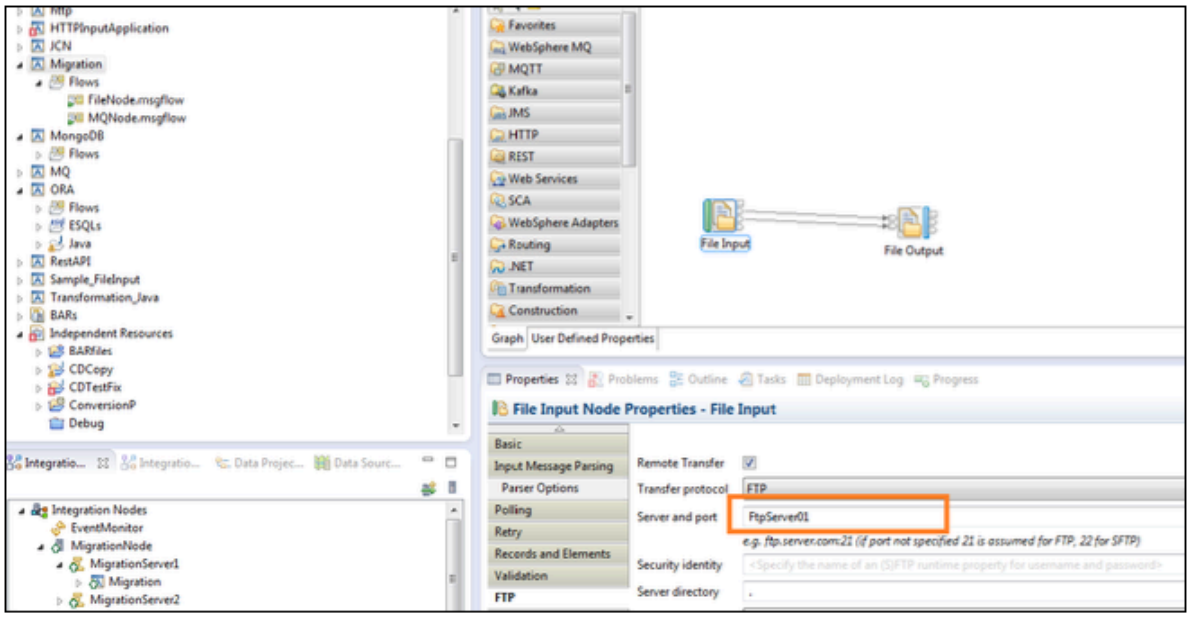
8. Start the independent integration server by using the **`IntegrationServer`** command:

```
IntegrationServer --name MigrationServer1 --work-dir C:\temp\MigrationServer1
```

9. To demonstrate that the configurable services that were defined for the IBM Integration Bus 10.0 integration nodes are copied with the migrated integration server, complete the following steps.

   a) Create a configurable service.
   `mqsicreateconfigurableservice MigrationNode -c FtpServer -o FtpServer01 -n serverName,scanDelay,transferMode,connectionType,securityIdentity -v one.hursley.abc.com:123,20,Binary,ACTIVE,secId`

   b) Modify your message flow to use the FTP configurable service.



   c) Save the message flow and redeploy the application to the integration server `MigrationServer1`.

   d) Create a new backup by using the **`mqsibackupbroker`** command in IBM Integration Bus 10.0.

```
mqsibackupbroker MigrationNode -d C:\temp\
```

   e) In IBM App Connect Enterprise 12.0, run the **`mqsiextractcomponents`** command:

```
mqsiextractcomponents --source-integration-node MigrationNode --source-integration-server
 MigrationServer1 --backup-file C:\temp\MigrationNode_181114_125348.zip --target-work-
directory C:\temp\MigrationServer4 --default-application DefApp
BIP8071I: Successful command completion.
```

   The folder `C:\temp\MigrationServer4\run\DefaultPolicies` contains the FTP configurable service policy under the IBM App Connect Enterprise 12.0 independent integration server. The FtpServer01 configurable service is available as a new policy. The `FtpServer01.policyxml` file contains configurations that match your FTP Server configurations.

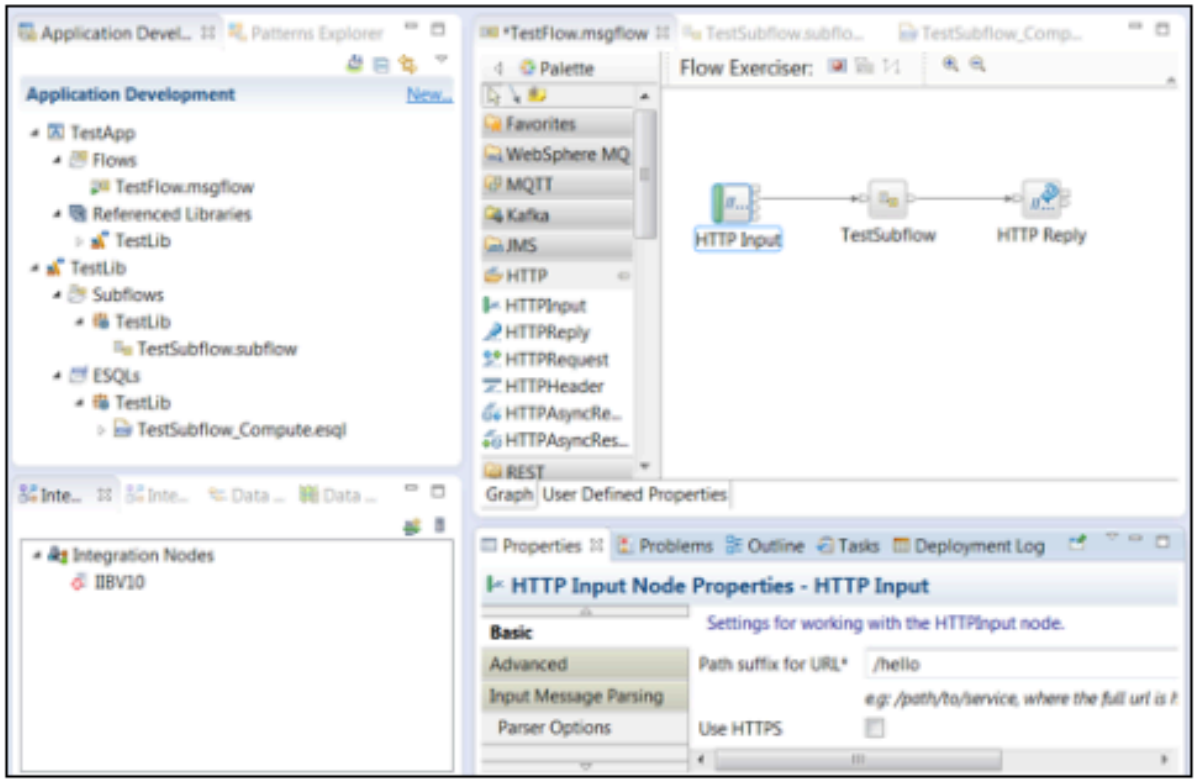
   f) Start the independent integration server by using the **`IntegrationServer`** command and verify the flow execution.

Migrating an integration server that contains applications and a shared library

Migrate an IBM Integration Bus 10.0 integration server that contains applications and a shared library to IBM App Connect Enterprise 12.0.

Procedure

1. In IBM Integration Bus 10.0, create a shared library to contain a simple subflow. Create an application with a simple message flow that uses the subflow from the shared library.




```
CREATE COMPUTE MODULE TestSubflow_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        -- CALL CopyEntireMessage();
        SET OutputRoot.XMLNSC.out = 'Hello ' || InputRoot.XMLNSC.in;
        RETURN TRUE;
    END;
```

2. Deploy the shared library to the IBM Integration Bus 10.0 integration node `IIBV10`.
3. Back up the IBM Integration Bus 10.0 integration node `IIBV10` by using the **`mqsibackupbroker`** command.

```
mqsibackupbroker IIBV10 -d c:\temp
```

4. Create a new IBM App Connect Enterprise 12.0 integration node called ACEV12 by using the command `mqsicreatebroker ACEV12`.
5. Start the IBM App Connect Enterprise 12.0 node ACEV12 to set up the work directory.
6. Stop the IBM App Connect Enterprise 12.0 node.
7. Create a blank work directory for the integration server `IS1` to be migrated to IBM App Connect Enterprise 12.0.

8. Use the **mqsiextractcomponents** command to migrate the IBM Integration Bus 10.0 integration server IS1 to IBM App Connect Enterprise 12.0, then start the integration node ACEV12.

```
mqsiextractcomponents --source-integration-node IIBV10 --source-integration-server IS1 --
backup-file c:\temp\IIBV10_181118_220025.zip
--target-work-directory c:\ProgramData\IBM\MQSI\components\ACEV12\servers\IS1
```

```
mqsistart ACEV12
```

9. After you start the node, connect to the integration node from the IBM App Connect Enterprise Toolkit.

   The application and shared library resources that were migrated from the IBM Integration Bus 10.0 node appear under the integration node ACEV12.

10. Test the migrated flow.
    For example:

```
C:\user\curl_7_53_1_openssl_nghttp2_x64>curl -d "<in>Test</in>" http://localhost:7800/hello
<out>Hello Test</out>
```

Migrating an SAP integration flow

Migrate an SAP integration flow that runs on IBM Integration Bus 10.0 to IBM App Connect Enterprise 12.0.

About this task

The following steps describe how to create an SAP integration flow in IBM Integration Bus 10.0, then how to migrate it to IBM App Connect Enterprise 12.0. The steps describe two different migration options: migrating to an integration node or an independent integration server. This scenario uses an SAP outbound adapter as an example; the same steps apply to an inbound adapter.

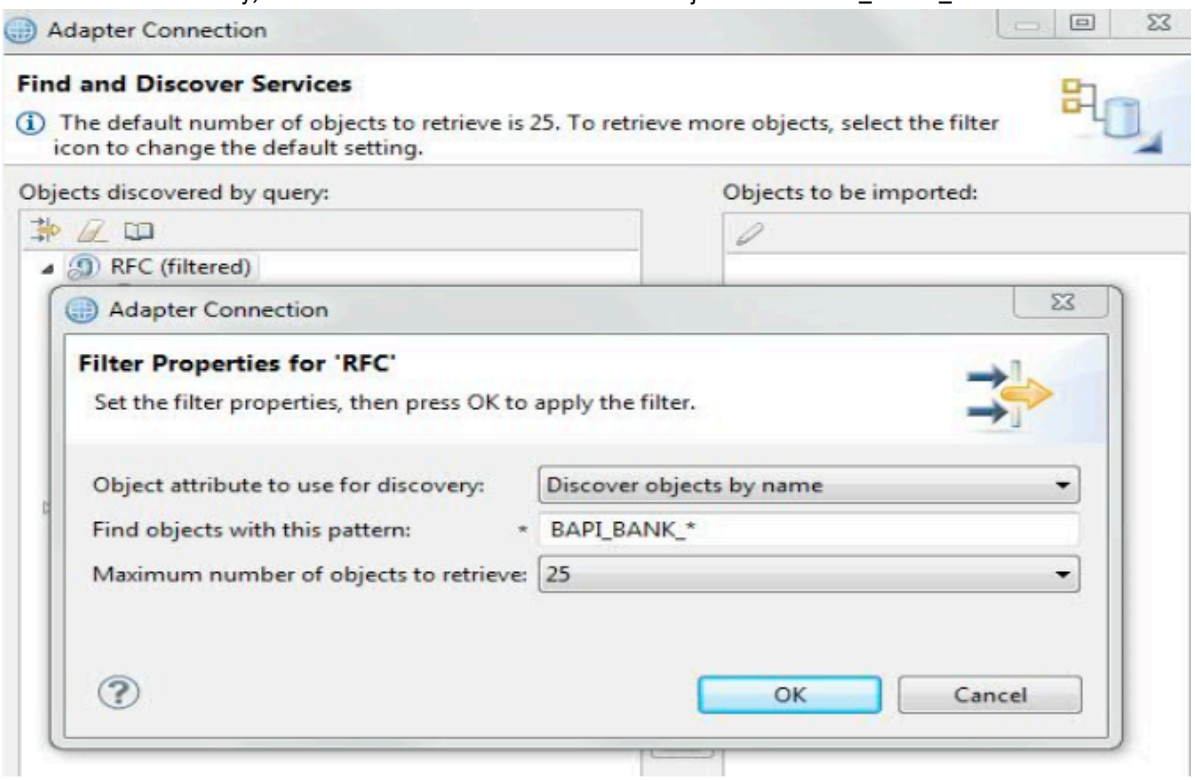
Procedure

- “Creating an SAP integration flow in IBM Integration Bus 10.0” on page 35
- “Migrating an IBM Integration Bus 10.0 integration node to an IBM App Connect Enterprise 12.0 integration node” on page 39
- “Migrating an integration flow from an IBM Integration Bus 10.0 integration node to an IBM App Connect Enterprise 12.0 independent integration server” on page 40

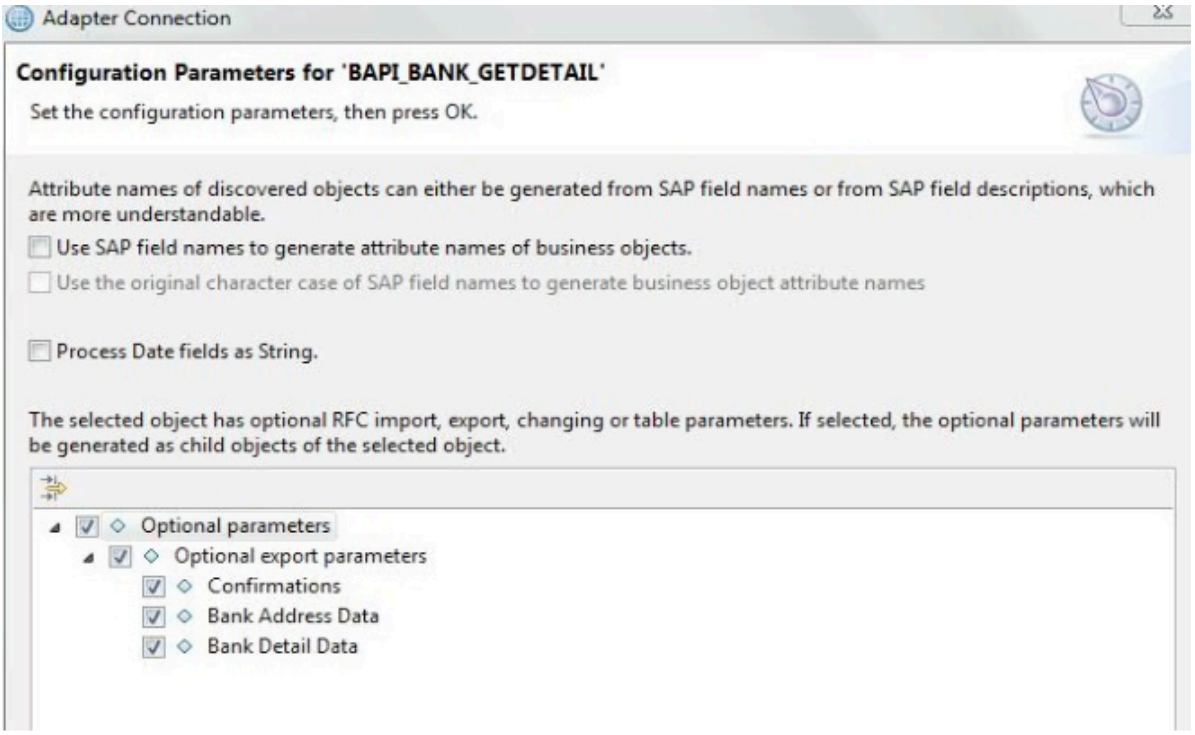
Creating an SAP integration flow in IBM Integration Bus 10.0

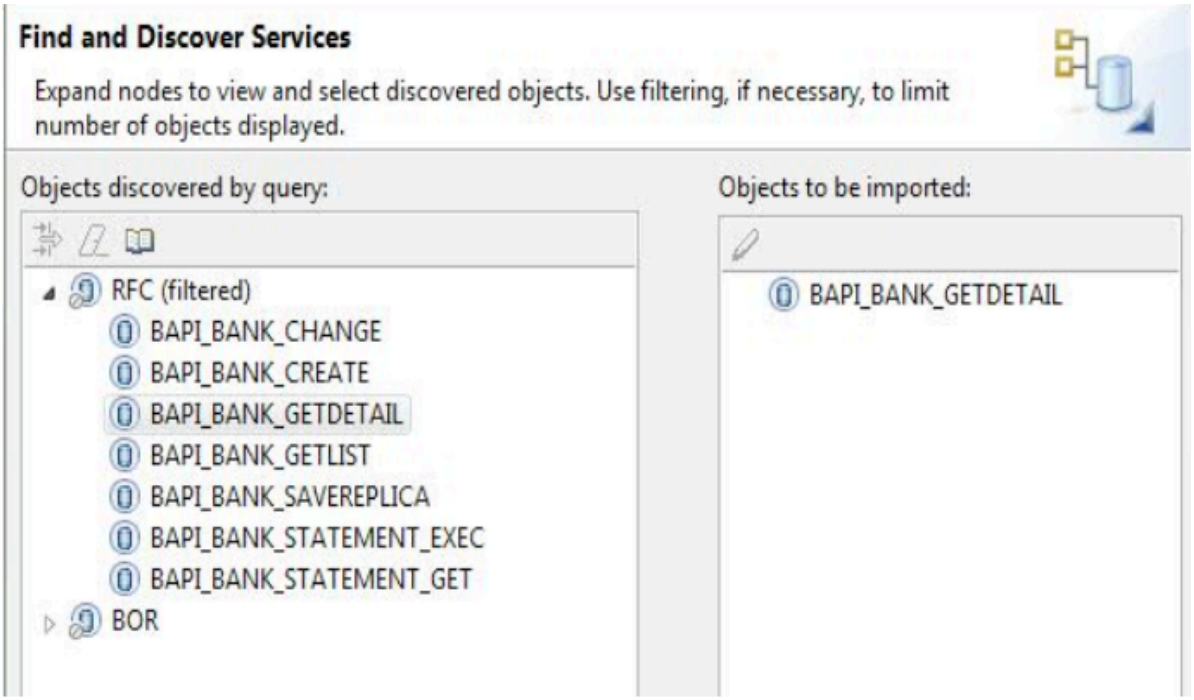
Procedure

1. In IBM Integration Bus 10.0, create an SAP outbound adapter to fetch `BAPI_BANK_GETDETAILS`.
   a) In the IBM Integration Toolkit, click **File** > **New** > **Adapter connection**, then follow the instructions in the wizard.
   b) For service discovery, select **RFC** and set a filter to find objects for `BAPI_BANK_*`.

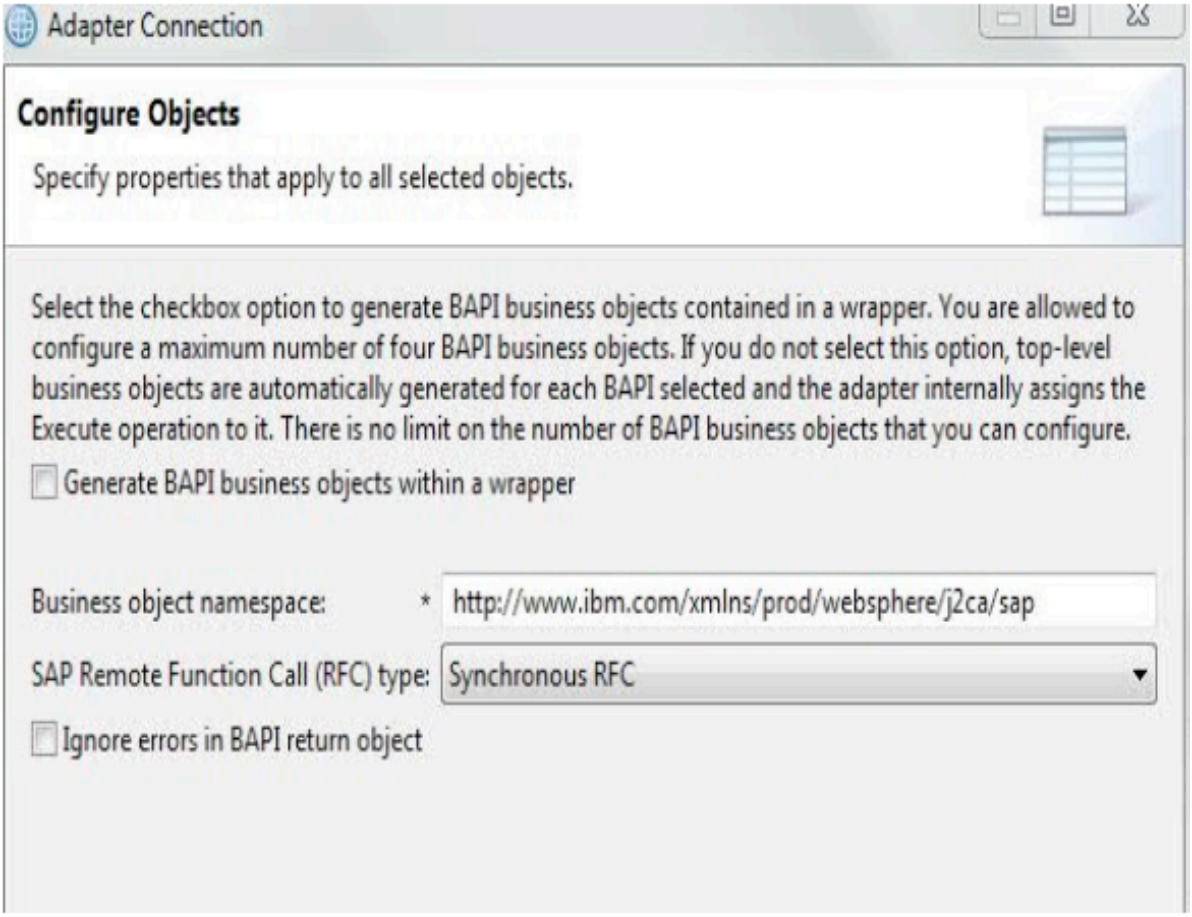


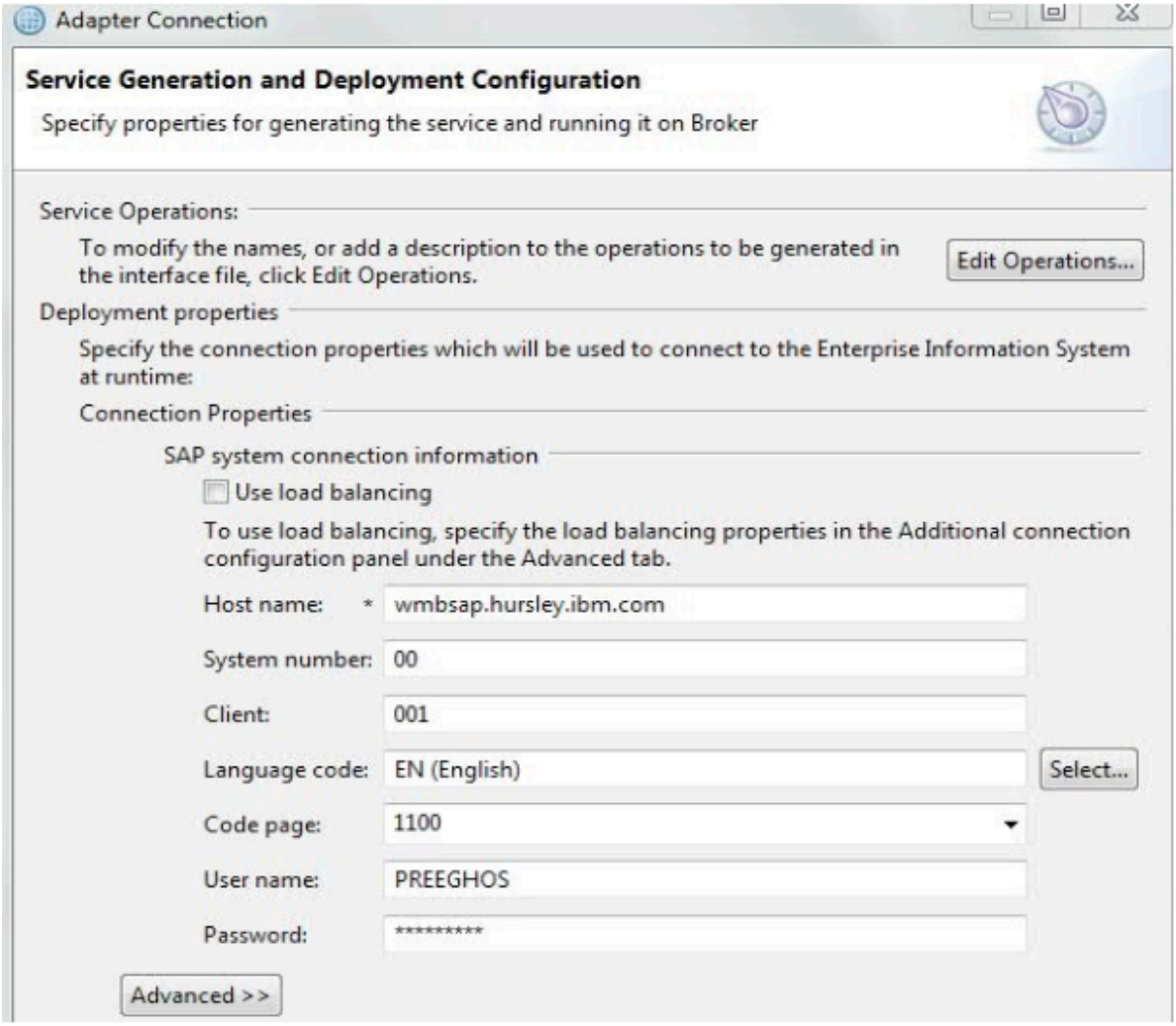
   c) For the the `BAPI_BANK_GETDETAIL` configuration parameters , select the objects to import.





d) Configure the objects.





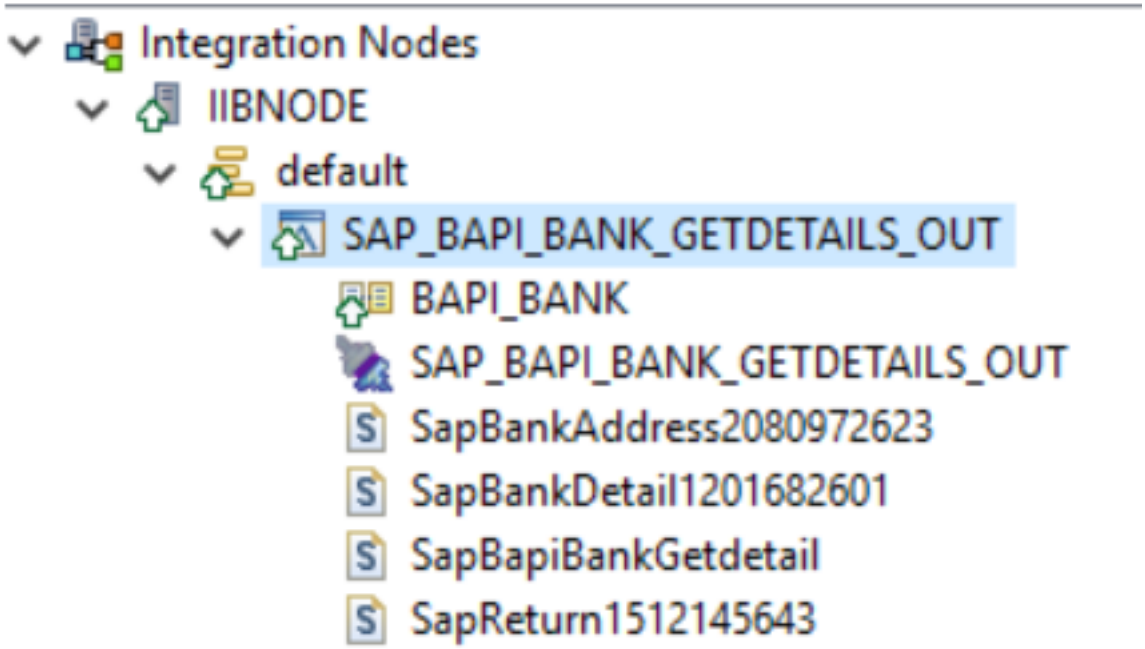
2. Create a simple SAP application to fetch `BAPI BANK GETDETAILS`.
3. Configure the integration node with SAP JCo libraries by using the **`mqsichangeproperties`** command.

```
mqsichangeproperties IIBNODE -c EISProviders -o SAP -n jarsURL -v C:\SAP_JARS
mqsichangeproperties IIBNODE -c EISProviders -o SAP -n nativeLibs -v C:\SAP_JARS
```

You can verify that the properties are set up correctly by running the command:
`mqsireportproperties IIBNODE -c EISProviders -o SAP -r`

```
C:\Program Files\IBM\IIB\10.0.0.16>mqsireportproperties IIBNODE -c EISProviders -o SAP -r

EISProviders
  SAP
    jarsURL='C:\SAP_JARS'
    nativeLibs='C:\SAP_JARS'

BIP8071I: Successful command completion.
```

4. Deploy the resources to the integration server.



5. Create a configurable service for the SAP adapters connection.

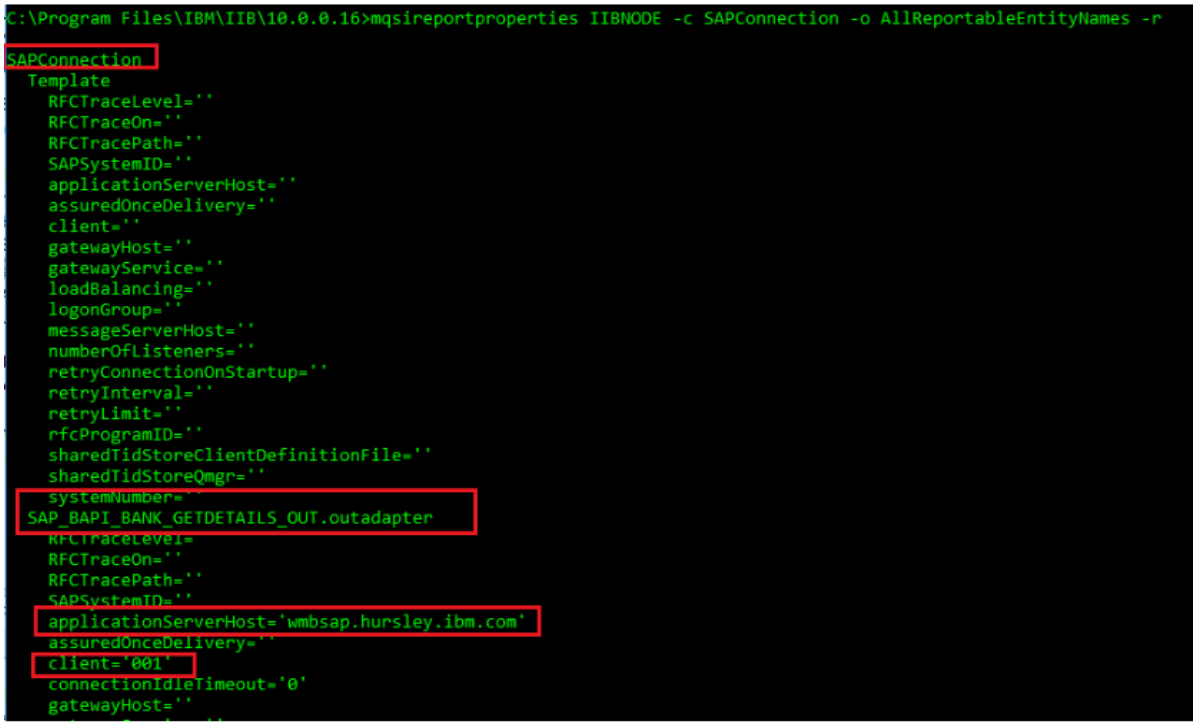
   By using a configurable service, you can change the host name that is set in the adapter file without needing to redeploy the resources.

   a) Use the SAPConnection configurable service to change connection details for an SAP adapter.

```
mqsicreateconfigurableservice IIBNODE -c SAPConnection -o
 SAP_BAPI_BANK_GETDETAILS_OUT.outadapter -n applicationServerHost,client -v
 wmbsap.hursley.ibm.com,001
```

   b) To display all SAPConnection configurable services, use the **mqsireportproperties** command:

```
mqsireportproperties IIBNODE -c SAPConnection -o AllReportableEntityNames -r
```



Migrating an IBM Integration Bus 10.0 integration node to an IBM App Connect Enterprise 12.0 integration node

Procedure

1. Back up the IBM Integration Bus 10.0 integration node by using the **mqsibackupbroker** command.

```
mqsibackupbroker IIBNODE -d C:\SAP_JARS
```

2. Migrate the IBM Integration Bus 10.0 integration node and resources to IBM App Connect Enterprise 12.0 by using the **mqsiextractcomponents** command.

```
mqsiextractcomponents --source-integration-node IIBNODE --target-integration-node ACENODE --
backup-file "C:\SAP_JARS\IIBNODE_190712_083247"
```

   This **mqsiextractcomponents** command creates a new integration node and integration server. If the integration node already exists, the **mqsiextractcomponents** command fails unless you

specify –delete-existing-node. In this case, the existing integration node is deleted and a new integration node is created with the same name.

All deployed resources are migrated under the /run directory of the created integration server. For example, on Windows: C:\ProgramData\IBM\MQSI\components\IntegrationNode\servers\IntegrationServer.

During migration, the IBM Integration Bus 10.0 configurable services are converted to policies in IBM App Connect Enterprise 12.0. You can find the policy project in the integration node work directory. If you do not specify a policy project name, policies are created in a policy project called DefaultPolicies. The policy file that is created has the same name as the adapter.

3. Start the IBM App Connect Enterprise 12.0 integration node by running the command mqsistart ACENODE.

   The application and policy files are listed under the IBM App Connect Enterprise 12.0 node in the Toolkit. You can confirm the location of the migrated JAR files by running the **mqsireportproperties** command on the IBM App Connect Enterprise 12.0 node.

```
mqsireportproperties ACENODE -c EISProviders -o SAP -r
```

4. Test the integration flow by sending a message to fetch the record for BAPI BANK GETDETAILS.

   The following example shows the expected output message.

```
<NS1:SapBapiBankGetdetail xmlns:NS1="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/sapbapibankgetdetail">
<SapBankAddress>
<NameOfBank>Laan og Spar</NameOfBank>
<HouseNumberAndStreet>Havnegade 6</HouseNumberAndStreet>
<City>1900 Copenhagen V</City>
<BankNumber>1200</BankNumber>
</SapBankAddress>
<SapBankDetail>
<DateOnWhichTheRecordWasCreated>1997-10-02</DateOnWhichTheRecordWasCreated>
<NameOfPersonWhoCreatedTheObject>SAP</NameOfPersonWhoCreatedTheObject>
</SapBankDetail>
<SapReturn>
<MessageNumber>000</MessageNumber>
<ApplicationLogInternalMessageSerialNumber>000000</ApplicationLogInternalMessageSerialNumber>
<LinesInParameter>0</LinesInParameter>
</SapReturn>
</NS1:SapBapiBankGetdetail>
```

Migrating an integration flow from an IBM Integration Bus 10.0 integration node to an IBM App Connect Enterprise 12.0 independent integration server

Procedure

1. Migrate an IBM Integration Bus 10.0 integration server to an IBM App Connect Enterprise 12.0 independent integration server by running the **mqsiextractcomponents** command.

```
mqsiextractcomponents --source-integration-node IIBNODE --source-integration-
server default --target-work-directory "C:\temp\ACESIS" --backup-file "C:\SAP_JARS
\IIBNODE_190712_083247.zip"
```

   If the work directory already exists, the **mqsiextractcomponents** command fails unless you specify –clear-work-directory. In this case, the configuration and resources are written to the work directory, and overwrite any data that might be present in the directory.

2. Start the IBM App Connect Enterprise 12.0 integration server by running the **IntegrationServer** command.

```
IntegrationServer --name default --work-dir C:\temp\ACESIS
```

All deployed resources are migrated to the `/run` subfolder in the work directory of the independent integration server.

During migration, the values that are set by using the **mqsichangeproperties** command in IBM Integration Bus 10.0 are added to the `server.conf.yaml` file in the `/overrides` subfolder in the work directory of the IBM App Connect Enterprise 12.0 integration server.

Also during migration, the IBM Integration Bus 10.0 configurable services are converted to policies in IBM App Connect Enterprise 12.0. You can find the policy project in the `/run` subfolder of the work directory for the integration server. If you do not specify the policy project name, policies are created in a policy project called `DefaultPolicies`. The name of the policy file that is created is the same as the adapter.

The application and policy files are listed under the IBM App Connect Enterprise 12.0 integration server in the Toolkit.

Performing pre-migration tasks

Perform the prerequisite tasks for migration to IBM App Connect Enterprise 12.0.

Procedure

Complete the following steps before you migrate your integration nodes or integration servers.

- Back up your resources by following the instructions in "Backing up resources before migration" on page 41.
- If your environment includes access to databases, create ODBC definitions for the databases and specify appropriate database drivers; see "Updating ODBC definitions when you migrate to IBM App Connect Enterprise 12.0" on page 42.
- Optional: If you plan to move your architecture to adopt containers, run the Transformation Advisor tool, as described in "Running the Transformation Advisor tool" on page 44.

What to do next

After you complete your pre-migration tasks, complete migration by following the instructions in "Migrating to IBM App Connect Enterprise 12.0" on page 57.

Backing up resources before migration

Back up your resources before you start to migrate to IBM App Connect Enterprise 12.0.

Before you begin

To plan your migration strategy, read "Preparing for migration" on page 19.

Procedure

Before you complete migration tasks, back up your current resources by completing the following steps.

1. Optional: If your message flows access user databases through an ODBC connection, back up the ODBC files that you use for these connections.

   Take a copy of these files and store them safely in a different location.

2. Back up your Toolkit workspace and resources, such as message flow files, message set definition files, Java files, ESQL files, mapping files, XML Schema files, and BAR files.
   - Export all the projects from your current Toolkit.
   - Archive your workspace resources.

     If you manage your workspace resources in a shared repository (for example, CVS), follow standard backup procedures for safeguarding versions. Create a version for storing Version 12.0 resources.

     If you maintain your workspace resources on a local or shared disk, copy your workspace directory to a different location.

Updating ODBC definitions when you migrate to IBM App Connect Enterprise 12.0

Before you migrate an integration server or integration node, create ODBC definitions for databases and specify appropriate database drivers for IBM App Connect Enterprise 12.0.

About this task

The database drivers that are supported by IBM App Connect Enterprise 12.0 might be at a later version than the drivers that are used by previous versions.

Follow the instructions that are provided for your operating system:

- "Updating ODBC definitions on Windows systems" on page 42
- "Updating ODBC definitions on Linux systems" on page 43

Updating ODBC definitions on Windows systems

Procedure

1. To change the ODBC connection definitions, open the **ODBC Data Source Administrator** window, then open the **System DSN** page.
2. For each Oracle and Sybase database that is accessed by the integration node, compare the ODBC driver against the entries that are listed in the following table, where *n* is the level of the installed fix pack (if appropriate).

DBMS	IBM App Connect Enterprise 12.0 ODBC driver name
Sybase	IBM App Connect Enterprise 12.0.1.*n* - DataDirect Technologies 64-BIT Sybase Wire Protocol
Oracle	IBM App Connect Enterprise 12.0.1.*n* - DataDirect Technologies 64-BIT Oracle Wire Protocol

   If the ODBC driver does not match, associate the data source name with the new ODBC driver by completing the following steps.

   a) Delete the data source by clicking **Remove**.

   b) Re-create the data source with the new ODBC driver by clicking **Add**.

3. To change the XA resource manager definitions, complete the following steps.
   a) Open the **Properties** window of the integration node queue manager by using the IBM MQ Services snap-in.
   b) Open the **Resources** page.
   c) For each Oracle and Sybase database that participates in a global unit of work that is coordinated by the integration node queue manager, change the contents of the **SwitchFile** field.

      The following table specifies the entries that you must change for each database management system (DBMS). *install_dir* represents the fully qualified path name of the directory in which you installed IBM App Connect Enterprise.

DBMS	Original entry	New entry
Oracle	*install_dir*\server\bin\ukor8dtc22.dll or *install_dir*\server\bin\ukor8dtc23.dll or *install_dir*\server\bin\ukora24.dll or *install_dir*\server\bin\ukora26.dll	*WBIMB*\bin\ukora95.dll
Sybase	*install_dir*\server\bin\ukase22.dll or *install_dir*\server\bin\ukase23.dll or *install_dir*\server\bin\ukase24.dll or *install_dir*\server\bin\ukase26.dll	*WBIMB*\bin\ukase95.dll

   d) For changes to the switch file configuration to take effect, restart the integration node queue manager.

Updating ODBC definitions on Linux systems

Procedure

1. To change the ODBC connection definition, create an ODBC definitions file by following the instructions in "Connecting to a database from Linux and UNIX systems by using the IBM Integration ODBC Database Extender" on page 189.

   Before you run the commands at the new service level, check that your ODBCINI environment variable points to the new file and not to the existing file. Check that the ODBCSYSINI environment variable is set to point to the directory that contains your odbcinst.ini file.

2. To change the XA resource manager definitions, edit the queue manager configuration file (`qm.ini`) of the queue manager that is associated with the integration node.

   The `qm.ini` file is in the directory `/var/mqm/qmgrs/`*`queue_manager_name`*, where *`queue_manager_name`* is the name of the queue manager that is associated with the integration node.

   In the *XAResourceManager* stanza for each Oracle and Sybase database that participates in a global unit of work that is coordinated by the integration node queue manager, change the entry for the switch file.

   The following table specifies the entries that you must change for each operating system and database management system (DBMS).

DBMS	Original entry	New entry

Oracle	`SwitchFile=UKor8dtc22.so`
	or
	`SwitchFile=UKoradtc22.so`
	or
	`SwitchFile=UKor8dtc23.so`
	or
	`SwitchFile=UKoradtc23.so`
	or
	`SwitchFile=UKoradtc24.so`
	or
	`SwitchFile=UKoradtc26.so`
		`SwitchFile=UKoradtc95.so`

Sybase (not supported on Linux on Z)	`SwitchFile=UKasedtc22.so`
	or
	`SwitchFile=UKasedtc23.so`
	or
	`SwitchFile=UKasedtc24.so`
	or
	`SwitchFile=UKasedtc26.so`
		`SwitchFile=UKasedtc95.so`

3. For changes to the switch file configuration to take effect, restart the integration node queue manager.

Running the Transformation Advisor tool

If you plan to move your architecture to adopt containers, run the Transformation Advisor tool before you migrate to IBM App Connect Enterprise 12.0. Use the tool to collect data about what is deployed to your IBM Integration Bus 10.0 or IBM App Connect Enterprise 11.0 integration node, then analyze the collected data for potential issues.

About this task

If you are planning to move to a containerized environment, the Transformation Advisor tool helps you to analyze your on-premises workloads for modernization. You can collect and assess data for a specific integration server in an integration node backup in App Connect Enterprise by using the **`TADataCollector`** command. You can also upload the results that are produced by the Transformation Advisor tool to IBM Cloud® Transformation Advisor. For more information, see IBM Cloud Transformation Advisor.

Procedure

To collect and analyze data about your IBM Integration Bus 10.0 or IBM App Connect Enterprise 11.0 integration node, complete the following steps.

1. Back up the resources that you want to migrate by using one of the following methods.
   - Create a backup file of your IBM Integration Bus 10.0 or IBM App Connect Enterprise 11.0 integration node by running the **mqsibackupbroker** command:

     ```
     mqsibackupbroker node10 -d C:\temp -a node10.zip
     ```

   - Add the resources that you want to migrate to a deployable BAR file by running the **mqsipackagebar** command:

```
mqsipackagebar -w C:\Workspace -a myapp.bar -c -i -k Application1 -y Shlib1 -v tracefile
```

2. Optional: In IBM App Connect Enterprise 12.0, create an output directory to write the logs from the **TADataCollector** command, and set the value of the environment variable *TADataCollectorDirectory* to the name of the output directory. If you do not create an output directory, the logs are written to a temporary folder in the home directory of the user who runs the command.
   a) Create an output directory by running the command `mkdir C:\TADemo`.
   b) Set the environment variable *TADataCollectorDirectory* to the path of the output directory by running the command `set TADataCollectorDirectory=C:\TADemo`.
3. In IBM App Connect Enterprise 12.0, run the **TADataCollector** command with one of the following options:
   - To collect data, run the command:

     ```
     TADataCollector ace collect C:\temp\node10.zip
     ```

   - To collect and assess data, run the command:

     ```
     TADataCollector ace assess C:\temp\node10.zip
     ```

   - To generate reports on data that is collected and assessed by the previous two options, run the command:

     ```
     TADataCollector ace report
     ```

   - To collect and assess data, and generate reports, run the command:

     ```
     TADataCollector ace run C:\temp\node10.zip
     ```

   For more information about the **TADataCollector** command, see command.

   When you run the **TADataCollector** command, output is written to an output directory. If you created the output directory and set the *TADataCollectorDirectory* environment variable, the output is written to that directory. Otherwise, the output is written to a temporary folder in the home directory of the user who runs the command. For example, `C:\Users\`*`username`*`\AppData\Local\Temp\TADataCollector`.

   The following subdirectories are created in the output directory:

   - `logs`: This folder contains the log *ta_util.log* that you can use to check for errors in the process.
   - `output`: This folder contains a file `environment.json`, which shows the details of the environment where the integration node was created. The folder also contains a subfolder for each integration server (for example, `node10/server10`), which contains a `.json` file. The `.json` file contains details of what is deployed to the integration server. It also contains configuration details that were generated when you ran the **TADataCollector** command.

4. Review the contents of the `output` folder in the output directory.

   If you ran the **`TADataCollector`** command with the `run` parameter, a static HTML report is produced; for example, `C:\TADemo\output\node10\recommendations.html`. The report lists any issues that are found for each integration server under the integration node.

   To view more detailed information about the issues, follow the links in the summary table or scroll down the page. Each issue has an *Overall Complexity Score* that denotes the type of action that is needed:

   - `Simple`: An administrative change is needed.
   - `Moderate`: A development change is needed.
   - `Complex`: A difficult development task is involved or an alternative strategy is needed.

   The Transformation Advisor tool also provides a severity classification for each issue it uncovers:

   - `Green (info)`: No immediate action is necessary, but you might want to be aware.
   - `Yellow (warning)`: Immediate action is probably needed or advised before you proceed.
   - `Red (error)`: You must take remedial action before you can proceed.

   The following example shows a complex issue with a severity classification of red that was identified when the tool was run against the IBM Integration Bus 10.0 integration node, *node10* with one integration server, *server10*.

Table 1. Recommendation report for assessment: node10

Assessment unit	Overall complexity score	Issues	Total effort (days)
server10	COMPLEX	1	10

Table 2. Recommendation report for assessment unit: server10

Product name	Product version	Runtime	Platform	Location
ACE	11.0	ACE	Docker	Private

Table 3. Overall complexity score: COMPLEX

RED issues: 1	Yellow issues: 0	Green issues: 0

Table 4. Issues with a COMPLEX complexity rating can be resolved by making significant development changes or by choosing an alternative technology:

ID	Title	Cost	Severity	Solution
IIB01	Consider a different transformation mechanism in place of .NET.	10	RED	The message flow contains an instance of a.NETInput or .NETCompute message flow node. While IBM App Connect Enterprise 12.0 software continues to support .NET, it does not support running the .NET CLR during deployment to Linux Docker containers on App Connect Enterprise Certified Containers. Other message flow nodes are available for transformation such as Compute, JavaCompute, and Mapping nodes.

For more information about the rules that apply to the Transformation Advisor tool, see “Rules for the Transformation Advisor tool” on page 47.

5. Complete any actions that are advised by the report, then run the Transformation Advisor tool again to confirm that all issues are resolved.

Rules for the Transformation Advisor tool

When you use the Transformation Advisor tool to analyze your deployed resources, different rules apply depending on whether you back up your resources to a backup file or a BAR file.

If you plan to move your architecture to adopt containers, you can run the Transformation Advisor tool to collect data about what is deployed to your IBM Integration Bus 10.0 or IBM App Connect Enterprise 11.0 integration node. You can then analyze the collected data for potential issues before you migrate. For more information, see “Running the Transformation Advisor tool” on page 44.

The following table shows the rules that apply to the Transformation Advisor tool and whether they apply to a backup file, a BAR file, or both.

Table 5. Rules for the Transformation Advisor tool:

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB001	Consider a different transformation mechanism in place of .NET.	True	True	The message flow contains an instance of a .NETInput or .NETCompute message flow node. While IBM App Connect Enterprise 12.0 software continues to support .NET, it does not support running the .NET CLR when it is deployed to Linux Docker containers on App Connect Enterprise Certified Containers. Other message flow nodes are available for transformation such as Compute, JavaCompute, and Mapping nodes.
IIB002	Consider a different transformation mechanism in place of PHP.	True	True	The message flow contains an instance of a PHPCompute message flow node. The PHPCompute node was deprecated in IBM Integration Bus 10.0 and it was removed from IBM App Connect Enterprise 11.0. Other message flow nodes are available for transformation, such as Compute, JavaCompute, and Mapping nodes.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB003	Consider an alternative mechanism to SCA to communicate with WebSphere Process Server.	True	True	The message flow contains an instance of an SCA message flow node (SCAInput, SCAReply, SCARequest, SCAAsyncRequest, SCAAsyncResponse). IBM App Connect Enterprise 12.0 does not support the SCA message flow nodes that were available in IBM Integration Bus 10.0. HTTP, IBM MQ, or JMS transport options are available for communication between message flows and SCA components in WebSphere Process Server.
IIB004	Consider a different mechanism to run IBM Operational Decision Management Business Rules.	True	True	The message flow contains an instance of a DecisionService message flow node. App Connect Enterprise v11.0.0.8 and later provides a replacement message flow node, the ODMRules node. The purpose of this node is to run ODM rules in the integration server. Alternatively, you can use the ODM SOAP or REST API to run business rules in the ODM engine.
IIB005	Consider a different protocol instead of relying on local file integration.	True	True	The message flow contains an instance of a FileInput message flow node that relies on local file interaction, and is not configured to use FTP. Although IBM App Connect Enterprise 12.0 continues to support local files with the FileInput node, if you want to use a container-based architecture, this choice has architectural drawbacks. Consider changing your configuration to use FTP or a more suitable messaging-based transport.
IIB006	Consider your use of WebSphere Service Registry and Repository.	True	True	The message flow contains an instance of a RegistryLookup or EndpointLookup message flow node, which communicates with WebSphere Service Registry and Repository. Releases before App Connect Enterprise 11.0.0.7 do not support these message flow nodes. If you have an urgent need, or a simple use case, you can use the SOAP nodes to communicate with WebSphere Service Registry and Repository. Alternatively, contact IBM to learn more about plans in this area.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB07	A Publication message flow node was found. You might want to consider altering your IBM MQ topology.	True	True	The message flow contains an instance of a Publication message flow node. In IBM Integration Bus 10.0 and in versions of IBM App Connect Enterprise 11.0 before Fix Pack 6, this message flow node requires a local server binding connection to an IBM MQ queue manager. IBM App Connect Enterprise 11.0 Fix Pack 6 and later supports the Publication node by using a remote client connection to an IBM MQ queue manager. Consider altering your IBM MQ topology as part of your move to a container-based architecture.
IIB08	A Sequence or Resequence message flow node was found. Consider altering your IBM MQ topology.	True	True	The message flow contains an instance of a Sequence or Resequence message flow node. In versions of IBM App Connect Enterprise 11.0 before Fix Pack 7, this message flow node requires a local server binding connection to an IBM MQ queue manager. Consider altering your IBM MQ topology as part of your move to a container-based architecture.
IIB09	A Collector message flow node was found. Consider altering your IBM MQ topology.	True	True	The message flow contains an instance of a Collector message flow node. In versions of IBM App Connect Enterprise 11.0 before Fix Pack 7, this message flow node requires a local server binding connection to an IBM MQ queue manager. Consider altering your IBM MQ topology as part of your move to a container-based architecture.
IIB10	A TimeoutControl or TimeoutNotification message flow node was found. You might want to consider altering your IBM MQ topology.	True	True	The message flow contains an instance of a TimeoutControl or TimeoutNotification message flow node. In versions of IBM App Connect Enterprise 11.0 before Fix Pack 7, these message flow nodes require a local server binding connection to an IBM MQ Queue Manager. Consider altering your IBM MQ topology as part of your move to a container-based architecture.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB11	An AggregateControl, AggregateRequest, or AggregateReply message flow node was found. You might want to consider altering your IBM MQ topology.	True	True	The message flow contains an instance of an AggregateControl, AggregateRequest, or AggregateReply message flow node. In versions of IBM App Connect Enterprise 11.0 before Fix Pack 7, these message flow nodes require a local server binding connection to an IBM MQ queue manager. Consider altering your IBM MQ topology as part of your move to a container-based architecture. In IBM App Connect Enterprise 12.0 the Group nodes are suitable for similar aggregation use-cases but they use in-memory queuing and have no IBM MQ dependency.
IIB12	A KafkaConsumer or KafkaProducer message flow node was found. You might want to consider changing the version of your Kafka broker.	True	True	The message flow contains an instance of a KafkaConsumer or KafkaProducer message flow node. In App Connect Enterprise v11.0.0.4 (and earlier fix packs), the product uses a Kafka client at version 0.10.0.1. In App Connect Enterprise v11.0.0.5 (and later fix packs), the product uses a Kafka client at version 2.20. You might want to change to a different client version when you consider the compatibility of your Kafka broker.
IIB13	A JDEdwardsInput, JDEdwardsRequest, PeopleSoftInput, PeopleSoftRequest, SiebelInput, or SiebelRequest message flow node was found. You might want to consider the version of your App Connect Enterprise installation.	True	True	The message flow contains an instance of a JDEdwardsInput, JDEdwardsRequest, PeopleSoftInput, PeopleSoftRequest, SiebelInput, or SiebelRequest message flow node. These message flow nodes are also supported in IBM App Connect Enterprise 12.0. If you used configurable service definitions with these message flow nodes in IBM Integration Bus 10.0, you might want to consider the introduction of JDEdwards, PeopleSoft, and Siebel policy types in IBM App Connect Enterprise 12.0.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB14	An SAPInput or SAPRequest message flow node was found. You might want to consider changing the configuration of your App Connect Enterprise installation.	True	True	The message flow contains an instance of an SAPInput or SAPRequest message flow node. These message flow nodes are also supported in IBM App Connect Enterprise 12.0, but IBM Integration Bus configurable services are replaced with policies. When you move to a container-based architecture, consider how to make the SAP JCo libraries available to your containers and the settings in `server.conf.yaml`.
IIB15	A deprecated graphical mapping message flow node was found. Convert this node to the newer Mapping node.	True	True	The message flow contains an instance of the old graphical mapping message flow node. This type of message flow node is no longer supported. You must convert message maps to graphical data maps. The Toolkit provides a conversion tool for this purpose.
IIB16	A TCPIPServer message flow node was found. Consider altering the configuration of your containers to open the TCPIP port.	True	True	The message flow contains an instance of a TCPIPServer message flow node. If you intend to use a non-default TCPIP port in your containers, consider changing your `values.yaml` file.
IIB17	A LoopBackRequest message flow node was found. Consider altering the configuration of your containers to support this node.	True	True	The message flow contains an instance of a LoopBackRequest message flow node. Your IBM App Connect Enterprise 12.0 installation provides Node Package Manager (NPM) to configure the Loopback Java™ script modules to support this node. When you move to a container-based architecture, consider your build pipeline and its abilities to configure these supporting files in your container.
IIB18	A WebSphere Transformation Extender or IBM Transformation Extender message flow node was found. Consider changing the configuration of your containers to support this node.	True	True	The message flow contains an instance of a WebSphere Transformation Extender or IBM Transformation Extender message flow node. IBM App Connect Enterprise 11.0 Fix Pack 4 (and later) supports the use of this type of message flow node, which is used with IBM Transformation Extender v10. When you move to a container-based architecture, consider this version information.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB19	An MQInput, MQOutput, or MQGet message flow node that uses server bindings to a queue manager was found. You might want to consider changing the node to use IBM MQ Client bindings when you move to containers.	True	True	You might want to consider changing to use IBM MQ client bindings when you move to containers so that you can use smaller containers. Independently scaling the integration servers in your architecture from your queue managers is easier to achieve if you use client bindings rather than server bindings.
IIB20	A Healthcare Pack artifact is deployed to this server.	True	True	A Healthcare Pack artifact is deployed to this server. The IBM Integration Bus Healthcare Pack is not supported in IBM App Connect Enterprise 12.0. In App Connect Enterprise v11.0.0.8 (and later), support is provided for applications in healthcare environments through IBM App Connect for Healthcare v5.0.0.0. Consider upgrading to App Connect Enterprise v11.0.0.8 or later and investigate the features that are provided by App Connect for Healthcare V5.0.0.0.
IIB21	A top-level message flow was found that originates in an integration project. These artifacts must be moved to the default application in IBM App Connect Enterprise 12.0.	True	True	A top-level message flow was found that originates in an integration project. When top-level resources are migrated by using the **mqsiextractcomponents** command, they are moved to the default application in IBM App Connect Enterprise 12.0. Ensure that you consider the groupings that you require for all top-level resources. The groupings are likely to involve the adoption of application projects instead of integration projects. Applications and libraries (which were first introduced in WebSphere Message Broker Version 8.0) provide a more efficient way to isolate and group message flows and their associated artifacts. While BAR files that contain top-level message flows can still be deployed to IBM App Connect Enterprise 12.0, these artifacts replace previously deployed default application content on each deployment. Consider the groupings of message flows in light of this change in iterative deployment behavior.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB22	A top-level resource was found that originates in an integration project. These artifacts are moved to the default application in IBM App Connect Enterprise 12.0.	True	True	A top-level resource was found. Top-level resources are moved to the default application when they are migrated to IBM App Connect Enterprise 12.0 by using the **mqsiextractcomponents** command. When you migrate to IBM App Connect Enterprise 12.0, consider your groupings for all top-level resources. The top-level resource that this rule detected is likely to be a dependency of a top-level message flow. Consider which message flows depend on this resource. Group the resource so that it continues to be available to the message flow when the flow is moved from its integration project to an application project. You can look for instances of rule IIB21, which detects top-level message flows that might have a dependency on the top-level resource that is highlighted by this rule.
IIB23	A SOAPInput or HTTPInput message flow node was found that is using the integration node-wide listener.	True	False	The message flow contains an instance of a SOAPInput or HTTPInput message flow node that is using the integration node-wide listener. When you run App Connect Enterprise in a container architecture, you do not use an integration node or the integration node-wide listener. Instead, you use an independent integration server with its own embedded HTTP listener.
IIB24	A configuration indicates the use of the Record and Replay feature.	True	False	Record and Replay is available in IBM App Connect Enterprise 12.0. While you can run this capability in a container-based architecture, it depends on a relational database and IBM MQ publications.
IIB25	A SOAPInput or HTTPInput message flow node that uses HTTPS was found.	True	True	The message flow contains an instance of a SOAPInput or HTTPInput message flow node that is using HTTPS. IBM App Connect Enterprise 12.0 uses TLSv1.2, and can also use TLSv1.3 for inbound HTTPS communications. Ensure that TLSv1.2, or TLSv1.3, are acceptable to your Business Partner applications.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB26	A globally coordinated message flow was found.	True	True	A globally coordinated message flow was detected. When you adopt a container-based architecture, you are unlikely to want to use globally coordinated message flows in your containers. You might want to reconsider your architecture to avoid global coordination, or to enable these flows to keep running outside containers.
IIB27	Configuration indicates the use of the embedded global cache feature.	True	False	The embedded global cache feature is available in IBM App Connect Enterprise 12.0. It is not advisable to use this capability to share information between integration servers in a container-based architecture. If you do use this capability, consider the placement and persistence of your catalog servers.
IIB28	Configuration indicates the use of the multi-instance high availability feature.	True	False	The multi-instance high availability feature for integration nodes is available in IBM App Connect Enterprise 12.0. You are unlikely to use this model to achieve high availability if you move to a container-based architecture. If you do use this model, consider the persistence and disk requirements.
IIB29	An MRM message set dictionary was detected.	True	True	An MRM message set dictionary was detected. These artifacts are supported for use in IBM App Connect Enterprise 12.0, but consider using DFDL message modeling technology instead.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB30	A message flow with user-defined properties was found.	True	True	A message flow with user-defined properties was detected. Message flows can continue to use user-defined properties when deployed to IBM App Connect Enterprise 12.0. If you are moving to a container architecture, you will probably not want to dynamically change the value of user-defined properties after deployment. Instead, when you change configuration data, you are likely to stop your container and restart it with the new configuration applied. In the unlikely event that you want to dynamically update message flow user-defined properties after deployment, an administrative API function is available in IBM App Connect Enterprise 12.0. In general, when you use container-based architectures, other methods for providing configuration to an independent integration server might be preferable, such as a user-defined policy.
IIB31	An IBM Integration Bus Activity Log configurable service was detected, which wrote to local files.	True	False	When you migrate to IBM App Connect Enterprise 12.0, IBM Integration Bus Activity Log configurable services are converted into Activity Log policy documents. When you move to a container-based architecture, you might want to reconsider your chosen output format for Activity Logging.
IIB32	An integration server was associated with an integration node that specified a product edition (that uses the **mqsimode** command) that is no longer available.	True	False	While artifacts can be carried forward, due to changes in licensing, not all IBM Integration Bus software editions have direct IBM App Connect Enterprise 12.0 equivalents. Check with your IBM representative to ensure that you move to the appropriate App Connect Enterprise edition and remain licensed correctly in future.

Table 5. Rules for the Transformation Advisor tool: (continued)

ID	Title	Can be found in a backup file	Can be found in a BAR file	Solution
IIB33	An IBM Integration Bus configurable service was detected that cannot be updated dynamically when it is converted to a policy in IBM App Connect Enterprise 12.0.	True	False	When you migrate to IBM App Connect Enterprise 12.0, IBM Integration Bus configurable services are converted into policy documents. Policy documents have several advantages, such as they can be created by using Toolkit templates and deployed in a BAR file. Some policies cannot be updated dynamically without the need to restart an integration server. This requirement does not typically apply in container-based architectures. However, depending on your use cases, you might want to consider this requirement when you migrate to IBM App Connect Enterprise 12.0.
IIB34	A message flow that uses an MQTT Server was found.	True	True	A message flow uses an MQTT Server because it contains either an MQTTPublish or MQTTSubscribe message flow node. The built-in MQTT Server that is provided by IBM App Connect Enterprise 12.0 is not turned on by default in a container.
IIB35	A message flow that uses an MQTTSubscribe node to monitor IBM Integration Bus events was found.	True	True	A message flow contains an MQTTSubscribe message flow node with a topic root of `IBM/IntegrationBus`. The presence of this node might mean that you have a message flow that is designed to monitor the product itself, and to take further action when data is received. You might want to review this design pattern when you move to a container-based architecture. IBM Integration Bus 10.0 can be configured to publish the following types of information to MQTT: operational events, admin events, business events, flow statistics, and resource statistics. IBM App Connect Enterprise 12.0 can be configured to publish the following types of information to MQTT: business events, flow statistics, and resource statistics. The App Connect Enterprise REST Administration API provides methods that can be called to provide operational and administration information.

Migrating to IBM App Connect Enterprise 12.0

Complete the tasks to migrate to IBM App Connect Enterprise 12.0.

Before you begin

- Plan your migration by reading the topics in “Preparing for migration” on page 19.
- Complete pre-migration tasks by following the instructions in “Performing pre-migration tasks” on page 41.

About this task

Note: IBM App Connect Enterprise 12.0 does not include IBM Integration Explorer; therefore, you cannot migrate IBM Integration Explorer. In IBM App Connect Enterprise 12.0, all integration node administration is done by using the web user interface, the IBM Integration API, or commands. For more information, see the following topics:

- web user interface
- #unique_95
- commands

Procedure

To migrate to IBM App Connect Enterprise 12.0, use one of the following methods.

- Use extract migration to migrate your integration servers and integration nodes by completing the steps in “Performing extract migration of an integration node or integration server” on page 57.

  You can use extract migration to migrate individual integration servers. Extract migration involves the extraction of configuration and resources from your source system to IBM App Connect Enterprise 12.0.
- Alternatively, you can migrate your integration nodes by using parallel migration, as described in “Performing parallel migration for an integration node ” on page 61.
- Migrate your development resources to IBM App Connect Enterprise 12.0 by following the instructions in “Migrating development resources to IBM App Connect Enterprise 12.0” on page 62.

Performing extract migration of an integration node or integration server

You can use the **mqsiextractcomponents** command to migrate from IBM Integration Bus 10.0 or IBM App Connect Enterprise 11.0 to IBM App Connect Enterprise 12.0. Use this command to migrate the configuration and resources of an integration node and of integration servers that are managed by an integration node.

Before you begin

- Plan your migration by reading the topics in “Preparing for migration” on page 19.
- Perform any pre-migration tasks by following the instructions in “Performing pre-migration tasks” on page 41.
- Find out about extract migration and the folder structure that it creates, by reading “Extract migration overview” on page 59.

About this task

Use the **mqsiextractcomponents** command to migrate integration nodes and managed integration servers. Independent integration servers that were created in IBM App Connect Enterprise 11.0 can be started in IBM App Connect Enterprise 12.0 without the need to migrate them or their resources.

However, integration server work directories are typically created as transient rather than long-term artifacts, and many users prefer to rebuild them when moving from one version to another.

Procedure

To complete extract migration for an integration node or a managed integration server, follow the appropriate set of steps:

- "Performing extract migration of an integration node" on page 58
- "Performing extract migration of a managed integration server" on page 58

Performing extract migration of an integration node

Procedure

1. On your source system, run the **mqsibackupbroker** command, specifying the name of the integration node that you intend to migrate to IBM App Connect Enterprise 12.0.

   The following example backs up integration node INODE on Windows, and saves it in the archive file `C:\MQSI\BACKUP\INODE.zip`:

   ```
   mqsibackupbroker INODE -d C:\MQSI\BACKUP -a C:\MQSI\BACKUP\INODE.zip
   ```

   For more information, see command.
2. Transfer the backup file that is created by the **mqsibackupbroker** command to an appropriate location.

   You can transfer the backup file to a location on the same system as the existing integration node, or to a location on a different system.
3. Run the **mqsiextractcomponents** command on the system to which you transferred the backup file.

   On the **mqsiextractcomponents** command, you must specify the following values:

   - The name of the backup file that is created when you run the **mqsibackupbroker** command on the source system
   - The name in the backup file of the integration node from which you are migrating
   - The name of the integration node to which to write the extracted configuration and resources

   You can also specify one or more optional parameters on the **mqsiextractcomponents** command. For more information, see command.
4. Optional: If you plan to run the migrated integration node on the same system as the existing integration node, stop the existing integration node by running the **mqsistop** command before you start the migrated integration node. For more information, see command.
5. Optional: If you plan to run the migrated integration node on the same system as the existing integration node, and with the same name as the existing integration node, delete the existing integration node by running the **mqsideletebroker** command before you start the migrated integration node. For more information, see command.
6. Start the migrated integration node by using the **mqsistart** command. For more information, see command.
7. After the **mqsistart** command completes, check the syslog, or the Windows Event log to confirm that the integration node has started successfully with no errors.

Performing extract migration of a managed integration server

Procedure

1. On your source system, run the **mqsibackupbroker** command, specifying the name of the integration node that is associated with the integration server that you intend to migrate to IBM App Connect Enterprise 12.0.

2. Transfer the backup file that is created by the **mqsibackupbroker** command to an appropriate location on your Version 12.0 system.
3. Run the **mqsiextractcomponents** command on your Version 12.0 system.

   On the **mqsiextractcomponents**, you must specify the following values:

   - The name of the backup file that is created when you run the **mqsibackupbroker** command on the source system
   - The name of the integration node with which the integration server that you are migrating is associated
   - The name of the integration server that you are migrating
   - The work directory to which the integration server components are to be extracted

   You can also specify one or more optional parameters on the **mqsiextractcomponents** command. For more information, see command.
4. When the **mqsiextractcomponents** command completes successfully, start the integration server on your Version 12.0 system by using the **IntegrationServer** command.

What to do next

When you complete migration, see the tasks in "Performing post-migration tasks" on page 64 for information about tasks that you might want to complete after migration.

Extract migration overview

You can use the **mqsiextractcomponents** command to migrate the configuration and resources of integration servers and integration nodes from your source system to your IBM App Connect Enterprise 12.0 system. The command extracts the configuration and resources from your source system and re-creates them in a directory structure on your target system. You can extract the configuration and resources from an integration node, including the configuration and resources from all of the integration servers that are associated with the integration node, or from individual integration servers on your source system.

The following information describes the folder structure that is created when you use extract migration on an integration node or an integration server.

Extract migration of an integration node

The extract migration of the configuration of an integration server or integration node from an earlier version results in the re-creation of the integration server or integration node on your IBM App Connect Enterprise 12.0 system. A node folder is created in the installation directory, or in the directory in which you have your registry. The resources that comprise the integration node configuration on your source system are put into the node folder. The configuration of all the integration servers that are associated with the integration node is also extracted, resulting in the re-creation of these integration servers on your Version 12.0 system. A server sub-folder is created in the node folder for each integration server. The resources that comprise each integration server configuration on your source system are put into the corresponding server sub-folder.

The following table summarizes the folder structure that is created when you perform extract migration of an integration node.

Table 6.

Folder	Sub-folder	Sub-folder	Sub-folder	Contents/description
<node name>				`node.conf.yaml` file. This file contains the settings for your migrated integration node. You can override the settings in the `node.conf.yaml` file by using the **mqsichangeproperties** command.

Table 6. (continued)				
Folder	**Sub-folder**	**Sub-folder**	**Sub-folder**	**Contents/description**
	`servers`			Sub-folders for every integration server that is server associated with the integration node. One sub-folder exists for each integration server.
		<server name>		Sub-folders containing the integration server configuration and resources. The name of the sub-folder is the same as the name of the integration server.
			`run`	Deployed resources.
			`overrides`	`server.conf.yaml` file. This file contains the settings for the integration server. You can override the settings in the `server.conf.yaml` file by using the **mqsichangeproperties** command.
	`policies`			Sub-folders for every policy that is created from a configurable service on the integration node on the source system.
		`NodePolicies`		
			policy_name`.policy.xml`	One policy file per migrated configurable service. The file contains the policy information.

Extraction of configuration and resources of an integration server

The extract migration of the configuration of an integration server from an earlier version results in the re-creation of an independent integration server on your IBM App Connect Enterprise 12.0 system. You might want to migrate an integration server in this way if you are operating in a container-based environment. The resources that comprise the integration server configuration are put into a work directory on your Version 12.0 system. You specify the work directory when you use the **mqsiextractcomponents** command.

The following table summarizes the directory structure that is created when you perform extract migration of an integration server.

Table 7.		
Folder	**Sub-folder**	**Contents/description**
<work_directory_name>		`server.conf.yaml` file. This file contains the settings for your migrated integration server. You can override the settings in the `server.conf.yaml` file by using the **mqsichangeproperties** command.
	`run`	Deployed resources, including policy projects, libraries, and applications.
	`overrides`	`server.conf.yaml` file. This file exists if you dynamically override settings in the integration server `server.conf.yaml` by using the **mqsichangeproperties** command.

Table 7. (continued)		
Folder	**Sub-folder**	**Contents/description**
	`log`	Files containing event log and syslog entries. By default, file names are of the format `integration_server.`*`integration_server_name`*`.entries.txt.`*`n`* where *n* is an integer in the range 1 through 9. When the integration server generates log entries, the most recent log data is held in a file that does not have the *n* suffix. As the data ages, it is held in files with the *n* suffix where the value of *n* increments with the age of the log data: the oldest data is held in the file that is suffixed 9. The files fill in an incremental fashion; when one file becomes full, entries are written to the next consecutive file.
	`config`	Files containing other configuration information such as security configuration.

To complete extract migration, follow the instructions in "Performing extract migration of an integration node or integration server" on page 57.

Performing parallel migration for an integration node

You can use parallel migration to do a staged migration process by creating a new Version 12.0 integration node to run in parallel with your existing integration node. You can then migrate your application logic from your existing integration node to your new integration node.

Before you begin

- Plan your migration by reading the topics in "Preparing for migration" on page 19.
- Complete pre-migration tasks by following the instructions in "Performing pre-migration tasks" on page 41.
- If your deployment uses functions that integrate with IBM MQ, or requires that the integration node has a specified queue manager, make sure that you have a supported version of IBM MQ. For more information, see "Installing IBM MQ" on page 96.
  - If you are reusing an existing IBM MQ deployment, migrate your queue managers and other resources to a supported version of IBM MQ by following the instructions in the IBM MQ product documentation.

    Note: If you are reusing an existing IBM MQ deployment, you can configure the new IBM App Connect Enterprise 12.0 integration nodes to use the same queue managers as the existing integration nodes. You do not have to create new queue managers for the new integration nodes.
  - If you are installing a new IBM MQ deployment, create one or more queue managers to support the necessary configuration for your IBM App Connect Enterprise 12.0 integration nodes.
- Check that you have no aggregations in progress on this integration node. When you migrate an integration node to Version 12.0, all live data that is being stored for aggregations in progress is lost.

Procedure

1. Install IBM App Connect Enterprise 12.0 on the same computer as your existing version, or on a different computer.

   If you are installing on the same computer, you must specify a different location.
2. Migrate your development resources to the IBM App Connect Enterprise Toolkit Version 12.0 (see "Migrating development resources to IBM App Connect Enterprise 12.0" on page 62).

3. Set up the correct Version 12.0 command environment for your operating system.
   - Linux On Linux systems, open a new shell and run the environment profile **mqsiprofile** for this Version 12.0 installation.
   - Windows On Windows, click **Start**, and open the Command Console that is associated with this Version 12.0 installation.
4. Create a Version 12.0 integration node by using the **mqsicreatebroker** command or the IBM App Connect Enterprise Toolkit. Give the integration node a name that is different from the name of the existing integration node.

   If any of the resources on the existing integration node require IBM MQ, specify a suitable queue manager for the new integration node (see "Installing IBM MQ" on page 96).
5. Start the Version 12.0 integration node by using the **mqsistart** command.

   You can also start a local integration node by using the IBM App Connect Enterprise Toolkit.
6. Make a list of the integration servers that you have on the existing integration node, and create these same integration servers on the Version 12.0 integration node.

   Use the web user interface, the IBM App Connect Enterprise Toolkit Version 12.0, or the **mqsicreateexecutiongroup** command to complete this step.
7. Deploy the message flows, applications, and libraries that are in use by the existing integration node to the Version 12.0 integration node from the Version 12.0 IBM App Connect Enterprise Toolkit.
8. Configure all other relevant properties of the existing integration node on the Version 12.0 integration node.
9. To delete your existing integration node, complete the following steps:
   a) In a command environment for your previous version, stop the original integration node by using the **mqsistop** command.
   b) Remove the original integration node from the IBM App Connect Enterprise Toolkit.
   c) In a command environment for your previous version, delete the original integration node by using the **mqsideletebroker** command.

What to do next

When you complete migration, check whether you need to complete any post-migration tasks in "Performing post-migration tasks" on page 64.

Migrating development resources to IBM App Connect Enterprise 12.0

You cannot migrate the IBM Integration Toolkit from previous versions, but you can install IBM App Connect Enterprise 12.0 to coexist with a previous version. You can then migrate the development resources to the IBM Integration Toolkit Version 12.0 by using a project interchange file.

Before you begin

Be aware of the following restrictions that apply after you migrate resources to IBM App Connect Enterprise Toolkit Version 12.0.

- You cannot share the development resources with previous versions of the IBM Integration Toolkit. When you create a new project in IBM App Connect Enterprise Toolkit Version 12.0, it is created in

a format that cannot be used in previous versions of the IBM Integration Toolkit. You can take the following steps to manage development with different versions of the IBM Integration Toolkit.

  - If you are using a version control system, create a new stream for use with the new version of your project.
  - If you expect to continue development of a project for an integration node at a version before Version 12.0, you must retain an IBM Integration Toolkit at the previous version.
  - If you need to continue development of the same project for integration nodes on both Version 12.0 and a previous version, ensure that you use the IBM Integration Toolkit from the previous version for development.
- You cannot deploy resources from IBM App Connect Enterprise Toolkit Version 12.0 to integration nodes at a previous version to Version 12.0.
- You can use the IBM Integration Explorer view of the IBM App Connect Enterprise Toolkit to add connections for integration servers and integration nodes. IBM App Connect Enterprise Toolkit Version 12.0 supports Version 12.0 integration servers and integration nodes only. Connections to integration servers and integration nodes that are running on previous releases are not supported.
- If any of the following actions cause an error in the project, the IBM App Connect Enterprise Toolkit marks the error, and you can use a Quick Fix to rectify the error.
  - Creating the metadata information for the user-defined node project
  - Correcting the plug-in identifier if it does not match the project name
  - Ensuring all the user-defined nodes are in the same category

  For more information, see "Applying a Quick Fix to a task list error" on page 2318. However, if you have different user-defined nodes that depend on different resources that have identical names, the BAR file compiler produces an error that indicates a naming conflict in the dependent resources.

About this task

You can continue to use resources from a previous version of IBM Integration Bus by importing them into an IBM App Connect Enterprise Toolkit Version 12.0 workspace. After you import resources, you can no longer use them in a previous version of the IBM Integration Toolkit.

Procedure

1. In the previous version of the Toolkit, export the resources that you want to use in IBM App Connect Enterprise Toolkit Version 12.0.
   a) Click **File** > **Export**.
      The **Export** wizard opens.
   b) Expand **Other**, click **Project Interchange**, then click **Next**.
   c) Select the projects that you want to export, and specify the location for the compressed file that is created. You can save the file to a folder on your file system, or to a disk drive.
   d) Click **Finish**.
      The project interchange file is created in the specified location.
2. Create an IBM App Connect Enterprise Toolkit Version 12.0 workspace, or open an existing one.
3. Import the project interchange file into the Version 12.0 workspace.
   a) Click **File** > **Import**.
      The **Import** wizard opens.
   b) Expand **IBM Integration**, click **Project Interchange**, then click **Next**.
   c) Specify the location of the project interchange file that you created in step 1.
   d) Specify the location of the open Version 12.0 workspace.
   e) Select the projects that you want to import into your Version 12.0 workspace, then click **Finish**.

Performing post-migration tasks

After you migrate to IBM App Connect Enterprise 12.0, finish setting up your environment.

About this task

Test the IBM App Connect Enterprise 12.0 integration node and integration server resources and components to verify that they are all present and working correctly. Some changes in behavior might be caused by defects that were fixed between versions.

The following topics describe tasks that you can complete after migration.

Procedure

- "Setting up a command environment" on page 64
- "Reconfiguring administration security after migration" on page 65
- "Setting up a global cache" on page 65
- "Migrating deployable resources" on page 66

What to do next

After you complete migration, use the product documentation for the previous version to complete the following tasks.

- Delete the components from the previous version.
- Remove the installed code for the previous version.

Setting up a command environment

Runtime components and commands that administer runtime components must be run from a command environment. You must initialize this environment by running the **mqsiprofile** command.

About this task

The **mqsiprofile** command places the Version 12.0 commands and libraries at the front of your search path, and can override any combination of PATH, CLASSPATH, or library PATH.

ODBC settings on Linux systems are found in a text file that is defined by the ODBCINI environment variable. If you are using Linux, set ODBCINI to point to a copy of the sample file *install_dir*/server/ODBC/unixodbc/odbc.ini, where *install_dir* is the IBM App Connect Enterprise installation directory.

On Linux, if you want to use IBM MQ features, you must set the IBM MQ environment where you want the integration server to run; for more information, see "Setting the IBM MQ environment on Linux and AIX" on page 84.

Ensure that you use this environment each time you run an administrative command, or start an integration server.

The **mqsiprofile** command also runs extra scripts that are copied to the common\profiles directory (on Windows) or the common/profiles directory (on Linux systems). These scripts can be used to initialize the environment for runtime components and other resources, such as databases.

The following steps explain how to initialize your command environment, either by explicitly running the **mqsiprofile** command (on Linux) or by starting the command console (on Windows).

Procedure

- On Windows, open a command console by searching for **IBM App Connect Enterprise Console**. If you have multiple installations of IBM App Connect Enterprise, make sure that you are running the IBM

App Connect Enterprise Console from the build of the IBM App Connect Enterprise installation that you want to administer.

- On Linux systems, locate and run the `mqsiprofile.sh` script in the directory in which you installed the appropriate product.

  ```
  . install_dir/server/bin/mqsiprofile
  ```

  You must include the period and space for this command to work correctly. If you want to run this command at the start of every session, add it to your login profile.

  If you use the `zsh` shell, running the **mqsiprofile** might cause the terminal session to exit. To resolve this issue, run the **unsetopt function_argzero** command before you run the **mqsiprofile** command.

What to do next

From the command line in the initialized environment, you can run commands to administer IBM App Connect Enterprise. For example, you can configure and start an integration server, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172 and "Starting an integration server" on page 250.

Reconfiguring administration security after migration

After migration, you can reconfigure your administration security to use file-based permissions that are set on your integration nodes.

About this task

In IBM Integration Bus 10.0, you can configure administration security by using queue-based permissions or file-based permissions on your integration nodes. In IBM App Connect Enterprise 12.0, you must use file-based permissions. For more information, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

Setting up a global cache

After migration of an integration node that is configured with a global cache, extra steps might be needed to complete the global cache configuration.

About this task

In IBM App Connect Enterprise 12.0, the global cache configuration involves setting the **cacheServerName** parameter, which must be unique in your global cache system.

If the name of the target integration node is the same as the name of the source integration node, no further configuration is needed. However, if the name of the target integration node is different from the name of the source integration node, make the following changes to complete the configuration of the global cache.

Procedure

- Define a unique **cacheServerName** in the `server.conf.yaml` files for the integration servers, and update the *cacheServerName* value in the **catalogClusterEndPoints** property.

  Consider this example: if the node `IB10NODE` is migrated from IBM Integration Bus 10.0 to IBM App Connect Enterprise 12.0 with the name `ACE12NODE`, the value of **catalogClusterEndPoints** is `IB10NODE_localhost_2800:localhost:2803:2801`. After migration, specify a value of, for example, `ACE12CatalogServer` in **cacheServerName** and update the value in

catalogClusterEndPoints to be ACE12CatalogServer:localhost:2803:2801, as shown in the following sample global cache stanza for a catalog server.

```
GlobalCache:
    cacheOn: true
    cacheServerName: 'ACE12CatalogServer'
    catalogClusterEndPoints: 'ACE12CatalogServer:localhost:2803:2801'
    catalogDomainName: 'WMB_IB10NODE_localhost_2800'
    catalogServiceEndPoints: 'localhost:2800'
    enableCatalogService: true
    enableContainerService: true
    enableJMX: true
    haManagerPort: 2801
    jmxServicePort: 2802
    listenerHost: 'localhost'
    listenerPort: 2800
```

Similarly, define a unique **cacheServerName** for the container server too. The value of **catalogClusterEndPoints** is the same for all the integration servers that participate in the global cache configuration. The global stanza for a container server, with a **cacheServerName** of ACE12ContainerServer2 looks like the following example:

```
GlobalCache:
    cacheOn: true
    cacheServerName: 'ACE12ContainerServer2'
    catalogClusterEndPoints: 'ACE12CatalogServer:localhost:2803:2801'
    catalogDomainName: 'WMB_IB10NODE_localhost_2800'
    catalogServiceEndPoints: 'localhost:2800'
    enableCatalogService: false
    enableContainerService: true
    enableJMX: true
    haManagerPort: 2805
    jmxServicePort: 2806
    listenerHost: 'localhost'
    listenerPort: 2804
```

Migrating deployable resources

You can continue to use existing resources in IBM App Connect Enterprise. However, if you want to continue developing resources created in previous versions, you must migrate them.

About this task

The following table summarizes the type of resource that you must migrate into IBM App Connect Enterprise 12.0 if you want to continue developing them.

Table 8. Existing resources that require extra migration tasks

Resource to continue developing in Version 12.0	**Resource that is created in Version 10.0**
Message sets	You must enable the menus for message set development (see “Migrating message sets” on page 67).
Message maps	You must complete the migration of message maps to graphical data maps for those existing message maps that you did not convert when you migrated to Version 10.0. New maps that are created in Version 10.0 are graphical data maps. See “Migrating message maps” on page 67.
Message flows	If your message flows include message flow nodes that are no longer available, rework your message flows to use available message flow nodes. See “Migrating message flows” on page 68.

Table 8. Existing resources that require extra migration tasks (continued)	
Resource to continue developing in Version 12.0	**Resource that is created in Version 10.0**
Subflows	You must complete the migration of the existing subflows that you did not convert when you migrated to Version 10.0. New subflows that are created in Version 10.0 are created as `.subflow` files. See "Migrating subflows" on page 68.
Custom integration applications	If you have applications that use the IBM Integration API, check that they access the correct resources. See "Migrating custom integration applications" on page 68.

Migrating message sets

About this task

In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134.

You can import message flows that contain message sets from previous versions into IBM App Connect Enterprise 12.0. Your existing message sets can be viewed, modified, and deployed in the usual way. However, by default, the menu options for creating new message sets or message definition files are hidden. To complete these tasks, you must first enable message set development in the IBM App Connect Enterprise Toolkit Preferences. For more information, see "Enabling message set development" on page 2248.

Migrating message maps

About this task

IBM App Connect Enterprise 12.0 includes a graphical data mapping capability, which is used when you add a Mapping node to a message flow.

You can import message flows containing the following nodes, which use message mapping into IBM App Connect Enterprise 12.0:

- DataDelete
- DataInsert
- DataUpdate
- Extract
- Mapping
- Warehouse

The message map (`.msgmap`) on these nodes can be viewed but they cannot be deployed to the runtime environment because they are not available in IBM App Connect Enterprise 12.0. The message-mapping operations from previous versions are accessible only in read-only mode and cannot be modified or deployed.

To deploy a message flow that uses an existing message map (`.msgmap` file), you must first replace the existing node with a new Mapping node. You must also convert the existing message map to a graphical data map that can be used by the new Mapping node (`.map` file). For more information, see Converting a message map from a `.msgmap` file a `.map` file.

Migrating message flows

About this task

The message flow nodes that are available in IBM App Connect Enterprise 12.0 might be different from the nodes that are available in the version from which you are migrating. If your message flows contain message flow nodes that are no longer available, you must rework them to use only those message flow nodes that are available at IBM App Connect Enterprise 12.0.

Migrating subflows

About this task

Since WebSphere Message Broker Version 8, you create subflows as `.subflow` files.

Subflows from earlier versions of the product were created as `.msgflow` files. You must convert them into `.subflow` files if you want to continue developing them in IBM App Connect Enterprise 12.0. You convert these subflows by using the function **Convert to subflow**. For more information, see "Converting between message flows and subflows" on page 576.

Migrating custom integration applications

About this task

Your existing Version 10.0 custom integration applications can be migrated to work with Version 12.0 integration nodes. However, if you are writing new applications, ensure that you use the `com.ibm.integration.admin.proxy.*` classes rather than the deprecated `com.ibm.broker.proxy.*`.

The IBM Integration API is no longer asynchronous because it is backed by synchronous REST calls. As a result, the `AdministeredObjectListener` framework is no longer available, and no `processModify()`, `processDelete()` callbacks exist. You must call refresh on the proxy objects to get the latest property values, such as for the deprecated Execution Group Proxy `myExecutionGroupProxy.refresh()`.

Procedure

1. For custom integration applications that connect to an integration node, update the connection details that are used by your custom integration applications to connect to the appropriate integration node.

   In IBM App Connect Enterprise 12.0, the IBM Integration API connections are no longer dependent on IBM MQ, and so connections are no longer made by using IBM MQ queues.
2. If you are connecting to a remote integration node, update your connection details to use the web administration port. For more information, see "Configuring the IBM App Connect Enterprise web user interface" on page 87.

   For information about using SSL on these connections, see "Connecting to an integration node by using the Toolkit" on page 245. If you are connecting to a local integration node, you can use a direct local connection.
3. If administration security is enabled, in IBM App Connect Enterprise 12.0, user credentials are needed by integration applications that use a remote connection. If the connection is local, you must run the application under a user ID that is part of the mqbkrs group. For new integration applications, you can provide user details during application development. For existing compiled applications, you must supply the security credentials by using one of the methods that is described in "Connecting to an integration node from a custom integration application" on page 447.

Chapter 3. Installing and uninstalling IBM App Connect Enterprise

Install and uninstall IBM App Connect Enterprise and complementary products.

About this task

Use the following topics to learn how to install and uninstall IBM App Connect Enterprise and optional complementary products on all supported platforms.

Installing IBM App Connect Enterprise

Review the tasks that are involved in the installation of IBM App Connect Enterprise.

About this task

The tasks that you perform to complete the installation are listed in the following steps; each task indicates whether it is required or optional. A summary of each task is provided, along with pointers to topics or sections that describe the task in more detail.

Service updates and other fixes are delivered occasionally in the form of fix packs. You can find information about available fix packs on the IBM App Connect Enterprise support web page. IBM App Connect Enterprise fix packs are complete App Connect Enterprise installations and are installed in the same way as the initial product release. You can install a fix pack on the same computer as a previous release, or on a different computer; for information about installing more than one fix pack (at the same or different levels) on a single computer, see “Coexistence of IBM App Connect Enterprise 12.0 with previous versions” on page 70. For more information about applying service updates, see “Applying service to IBM App Connect Enterprise” on page 90.

Procedure

1. Required: Make sure that you have acquired the product packages that you need for installation.

   Electronic images of IBM App Connect Enterprise are available from the IBM Passport Advantage® website.
2. Required: Make sure that you have access to the product documentation that you need for installation; see “Finding the latest information” on page 1.
3. Required: Determine the platforms on which to install IBM App Connect Enterprise.

   On Windows and Linux, the IBM App Connect Enterprise installation includes IBM App Connect Enterprise Toolkit. For more information, see #unique_127.
4. Required: Prepare each computer on which you are installing IBM App Connect Enterprise.
   a) Required: Check that your target computers meet the initial hardware, storage, and software requirements.

      For more information, see the IBM App Connect Enterprise system requirements web page.
   b) Required: Understand the security requirements for running IBM App Connect Enterprise; see “Security requirements” on page 72.
   c) Optional: Configure temporary directory space to enable IBM App Connect Enterprise Java components to operate correctly; see “Configuring temporary space on distributed systems” on page 74.
   d) Optional: If you are installing IBM App Connect Enterprise on Linux or AIX, check your kernel configuration; see “Checking the kernel configuration on Linux and AIX” on page 75.

5. Required: Install IBM App Connect Enterprise; see “Installing IBM App Connect Enterprise software” on page 75.
6. Optional: If required, install one or more language packs for IBM App Connect Enterprise Toolkit; see Installing language packs for the IBM App Connect Enterprise Toolkit.
7. Optional: Set up a command environment so that you can run administrative commands; see “Setting up a command environment” on page 64.
8. Optional: If required, install any complementary products; see “Installing and uninstalling complementary products” on page 96.

Results

You have reviewed the installation steps.

Coexistence of IBM App Connect Enterprise 12.0 with previous versions

IBM App Connect Enterprise 12.0 can coexist with IBM App Connect Enterprise 11.0 and IBM Integration Bus 10.0, so that you can test and compare functional differences between the versions.

You can control your migration to IBM App Connect Enterprise 12.0 by migrating individual integration nodes and integration servers; you do not have to migrate all integration nodes or servers at the same time. For more information, see Chapter 2, “Migrating,” on page 19.

The following sections describe how to achieve coexistence, and explain the restrictions that apply for each platform:

- “Coexistence on Windows” on page 70
- “Coexistence on Linux and AIX” on page 71

When you have an environment that contains different versions of IBM App Connect Enterprise and IBM Integration Bus, the following tasks have limitations:

Editing source artifacts in the IBM App Connect Enterprise Toolkit
Artifacts cannot be shared between different versions of the IBM App Connect Enterprise Toolkit and IBM Integration Toolkit. Changes made to an artifact by using the IBM App Connect Enterprise Toolkit make the artifact incompatible with earlier versions of the IBM Integration Toolkit. Consider branching your version control system when you migrate to a new version.

Building BAR files from artifacts
Your build system must use the same or later version as the IBM App Connect Enterprise Toolkit or IBM Integration Toolkit system that was used to create the artifact. A build system cannot build a BAR file from an artifact that was created or edited in a later version of the IBM App Connect Enterprise Toolkit or IBM Integration Toolkit.

Deploying BAR files to an integration node
You can deploy BAR files that are built with one version of IBM Integration Bus (by using the IBM Integration Toolkit, **`mqsicreatebar`**, or **`mqsipackagebar`**) to an integration node at a later version. You do not have to rebuild your BAR files. However it is not supported to deploy BAR files to an integration node at an earlier version. The BAR files are likely to contain new properties that cause deployment errors.

Coexistence on Windows

The following restrictions apply to coexistence on Windows.

- You can install multiple instances of IBM App Connect Enterprise on the same computer, but you cannot install multiple instances of a product version at the same modification or fix pack level. For example, you can install Version 12.0.2.0 on the same computer as Version 12.0.1.0, but you cannot install two instances of Version 12.0.1.0 on the same computer.
- All integration nodes from all versions are accessible by all users of the Windows system, and must have unique names.

- Each integration node is associated with the product code from a specific installation location and uses that product code when the integration node is started.
- If you have more than one installation on a single computer, ensure that the commands that you issue to integration nodes on that computer are using the correct version of installed code. On Windows, an IBM App Connect Enterprise Console is available for each installation. You must run commands in the correct instance of the IBM App Connect Enterprise Console for a particular installation.

  If you prefer, you can run the **mqsiprofile** command. By default, this command is stored in `C:\Program Files\IBM\ACE\12.0.n.0\server\bin`.

Coexistence on Linux and AIX

The following restrictions apply to coexistence on Linux and AIX:

- You can install multiple instances of IBM App Connect Enterprise and IBM Integration Bus on the same computer, including multiple instances of a product version at the same modification or fix pack level.
- Each integration node can be either part of a shared installation or part of a single-user installation. Integration nodes that are part of a shared installation are accessible by all users of the IBM App Connect Enterprise installation. Integration nodes that are part of a single-user installation are only accessible by a specific user.
- Integration nodes are not automatically associated with product code from specific installation locations. Before you start the integration node, you must run the **mqsiprofile** command that is associated with the product code that the integration node should use. The command is stored in *install_dir*`/ace-12.0.n.0/server/bin`, where *install_dir* is the directory where you unpacked the IBM App Connect Enterprise software. If you add the **mqsiprofile** command to your system login profile, it is run automatically whenever you log on.
- If you have more than one installation on a single computer, you must ensure that the commands that you issue to integration nodes on that computer are using the correct version of installed code. You must run the **mqsiprofile** command to set up the correct environment before you run other IBM App Connect Enterprise commands, such as **mqsicreatebroker**.

  If you installed an earlier version of this product on the same computer, check that the earlier **mqsiprofile** command is not included in the system profile for the current user ID. The environment that is set up by the **mqsiprofile** command of a previous version of IBM Integration Bus is not compatible with a later version of IBM App Connect Enterprise. To avoid potential problems, consider using a different user ID for each version and add the correct command for each version to the system profile of each user ID. For more information about the **mqsiprofile** command, see “Setting up a command environment” on page 64

Preparing the system

Complete the prerequisite tasks for your operating system before you install IBM App Connect Enterprise.

About this task

Before you install IBM App Connect Enterprise, read the topics that are relevant to your operating system:

- For Windows:
  1. “Security requirements” on page 72
  2. “Configuring temporary space on distributed systems” on page 74
- For Linux or UNIX systems:
  1. “Security requirements” on page 72
  2. “Configuring temporary space on distributed systems” on page 74
  3. “Checking the kernel configuration on Linux and AIX” on page 75

If you are using IBM MQ, configure the default settings for the log; see Default IBM MQ log settings.

For performance reasons, set the log size to the largest that the IBM MQ version that are using can support. If you are using a production system, consider using linear logging. By default, queue manager logs default to circular logging and use 3 x 1 MB log files.

What to do next

- On distributed systems, install IBM App Connect Enterprise; see "Installing IBM App Connect Enterprise software" on page 75.

Security requirements

Understand the security requirements before you install IBM App Connect Enterprise.

The security of IBM App Connect Enterprise components, resources, and tasks depends on the definition of users and groups configured in the operating system.

Some operating systems and other products impose restrictions on the structure and composition of user IDs:

- On Windows systems, user IDs can be up to 12 characters long, but on Linux and UNIX systems they are restricted to 8 characters. Database products, for example DB2®, might also restrict user IDs to 8 characters. If you have a mixed environment, ensure that the user IDs in the IBM App Connect Enterprise environment are limited to a maximum of 8 characters.
- Ensure that the case (upper, lower, or mixed) of user IDs in your IBM App Connect Enterprise environment is consistent. In some environments, uppercase and lowercase user IDs are considered the same, but in other environments, user IDs of different case are considered unique. For example, on Windows the user IDs 'tester' and 'TESTER' are identical, but on Linux and UNIX systems they are recognized as different user IDs.
- Check the validity of spaces and special characters in user IDs to ensure that, if used, these characters are accepted by all relevant systems and products in your IBM App Connect Enterprise environment.

If your user ID does not conform to these restrictions, you might have problems with installation or verification. If so, use an alternative user ID, or create a new one, to complete installation and verification.

Understand the security requirements that are associated with the operating systems that you are using:

- If you are installing on Linux or UNIX systems, see "Security on Linux and UNIX systems" on page 72.
- If you are installing on Windows, see "Security on Windows" on page 73.
-

When you have installed IBM App Connect Enterprise, check the *Security* topics in the IBM App Connect Enterprise product documentation to review and implement the security requirements for the performance of other tasks; see Chapter 8, "Security," on page 2491.

Security on Linux and UNIX systems

Understand the security requirements on Linux and UNIX systems before you install IBM App Connect Enterprise.

To deploy a single-user installation of IBM App Connect Enterprise on Linux or UNIX systems, you must have a user account on the system where you install IBM App Connect Enterprise. The user account does not need super user access to the system. After the deployment, the IBM App Connect Enterprise installation can be used by only you.

To deploy a shared installation of IBM App Connect Enterprise on Linux or UNIX systems, you must log in as root or as a super user who has write access to the `/var` directory. After the deployment, a security group that is called mqbrkrs is created (if this group does not already exist). To enable users to use the shared installation of IBM App Connect Enterprise, add the users to the mqbrkrs group by using the security facilities that are provided by your operating system; for example, the Systems Management Interface Tool (SMIT) on AIX, or the System Administration Manager on HP-Itanium.

Security on Windows

Understand the security requirements on Windows before you install IBM App Connect Enterprise.

To install IBM App Connect Enterprise on Windows, you must have the authority to install software on the system.

When you install IBM App Connect Enterprise, the installation wizard calls the **mqsisetsecurity** command, which completes the following tasks:

- Creates a security group called mqbrkrs.
- Adds your current user ID to the group mqbrkrs.

If you want to use IBM App Connect Enterprise with a different user ID from the ID that you use to install the product, you must add that user ID to the mqbrkrs group. Use either the Windows security facilities or the **mqsisetsecurity** command to complete this task; see command.

For an example that uses a Windows domain group topology to run IBM App Connect Enterprise in a Windows domain environment, see "Security in a Windows domain environment" on page 73.

Windows Terminal Services

If you are installing IBM App Connect Enterprise on a computer that runs Terminal Services, change the user mode to `install` before you start the installation, to ensure that actions taken during the installation are completed correctly; for example, that `.ini` files and other related files are created in the default system directory (`C:\Windows`). Restore the user mode to `execute` after the installation.

Security in a Windows domain environment

An example that uses a Windows domain group topology to run IBM App Connect Enterprise in a Windows domain environment.

About this task

You can use Windows domain groups to organize different levels of authorization to selective IBM App Connect Enterprise resources across your domain. To design and implement this domain group topology, add each domain group to the relevant local security groups on the domain workstations. You can now manage authorities by adding domain user accounts to the appropriate domain groups.

Procedure

1. Design your authorization group categories, and define domain groups on the domain controller system that correspond to these authorization categories, by using Windows security.
   For example, suppose that you have a single domain that contains three distinct sets of systems, which are used in development, testing, and production. Within your organization, various user roles require different levels of authorization to IBM App Connect Enterprise resources on those systems.

   Here is an example of how those authorization categories might map to domain groups:

Domain group	Description
ADM-MBprd	IBM App Connect Enterprise administrator authorities on production systems
ADM-MBuat	IBM App Connect Enterprise administrator authorities on test systems
ADM-MBdev	IBM App Connect Enterprise administrator authorities on development systems

2. Define and configure domain user accounts on the domain controller, by using Windows security. Add each domain user account to one or more domain groups to configure the access for that account. For example:

Table 9.

Domain user account	Role	Domain group membership
MBadmPRD	IBM App Connect Enterprise administrator for production systems	ADM-MBprd
MBadmUAT	IBM App Connect Enterprise administrator for test systems	ADM-MBuat
MBadmDEV	IBM App Connect Enterprise administrator for development systems	ADM-MBdev
john.smith	IBM App Connect Enterprise administrator for test and development systems	ADM-MBuat, ADM-MBdev

3. Install and configure IBM App Connect Enterprise on domain workstations.
   a) Install IBM App Connect Enterprise on the workstation.
   b) Add your domain groups to the local mqbrkrs group as appropriate.
      In this example, if a particular workstation is to serve as a development system, add the domain group `ADM-MBdev` to the local mqbrkrs group.

Configuring temporary space on distributed systems

Configure temporary directory space to enable IBM App Connect Enterprise Java components to operate correctly.

About this task

IBM App Connect Enterprise contains Java components. Some of these components require temporary directory space on the file system in order to extract class files from node packages (PAR files) and Java packages (JAR files).

The extracted files might not be visible to interactive users on the system, however, they consume space in the temporary directory.

The Java Runtime Environment also creates files in the temporary directory to support debug facilities.

To enable correct operation, allow at least 50 MB of space per integration server in the file system temporary directory for IBM App Connect Enterprise components. This space is required in addition to any space required for operation of user developed artifacts. Use of JavaCompute nodes, Java user-defined nodes, or additional plug-in nodes implemented in Java might require further temporary directory space.

On Linux, UNIX, and z/OS® computers, the TMPDIR directory is typically `/tmp`; on Windows computers, it is `c:\temp`. If this directory is not large enough to hold the JAR files, the integration node does not start.

To change the location of the temporary directory, use one of the following methods:

Procedure

- Use the environment variable `TMPDIR`.
- Set the system property `java.io.tmpdir`.

Checking the kernel configuration on Linux and AIX

Check the kernel configuration parameters on Linux and AIX for prerequisite and corequisite products.

About this task

IBM App Connect Enterprise has no specific requirements for kernel configuration parameters; however, other products might require particular settings. If you do not tune your kernel parameters to suit the products that you have installed, you might see unexpected results or a deterioration in performance.

Follow these steps to configure your kernel parameters:

Procedure

1. Check the documented values for the following products:
   - IBM MQ (if it is installed)
   - DB2 (if it is installed)
   - Other software that you have installed to work with IBM App Connect Enterprise, including other databases.

   You can access the relevant information for IBM products online at IBM Documentation.
2. Take the highest value for each parameter and compare it to the corresponding value in your kernel configuration.
3. If the current value is lower than the highest documented value, update the current setting by using the appropriate tooling that is supplied by the operating system provider. If the current value is higher, leave it unchanged.
4. If you have changed any kernel values, you might need to restart your system for these changes to take effect. Check the documentation for your operating system for further information about these parameters.

Installing IBM App Connect Enterprise software

You can install IBM App Connect Enterprise on a number of platforms.

About this task

To install IBM App Connect Enterprise, complete the following steps:

Procedure

1. Check the `readme.html` file for any updates to the installation instructions.

   For more details, see “Finding the latest information” on page 1.
2. Check that you have enough memory and disk space; see IBM App Connect Enterprise system requirements.
3. Depending on your operating system, complete the instructions to install IBM App Connect Enterprise:
   - 

See “Installing IBM App Connect Enterprise on Windows” on page 76.
   - 

See “Installing IBM App Connect Enterprise on Linux” on page 79.
   - 

See “Installing IBM App Connect Enterprise on AIX” on page 81.
4. Optional: To use IBM App Connect Enterprise Toolkit, in a supported language other than English; see Installing language packs for the IBM App Connect Enterprise Toolkit.

Installing IBM App Connect Enterprise on Windows

Install IBM App Connect Enterprise on Windows.

Before you begin

- Check the `readme.html` file for any updates to these installation instructions; see the product readme web page.
- Check that you have enough memory and disk space; see IBM App Connect Enterprise system requirements.
- Complete any prerequisite steps; see “Preparing the system” on page 71.

Installing the software

Procedure

To install IBM App Connect Enterprise, complete the following steps.

1. Extract the downloaded `.zip` file for the version of IBM App Connect Enterprise that you want to install to unpack the installation files into a local directory:
   For example, if you extract the file `ACESetup12.0.2.0.zip` into `C:\temp`, the following extracted files appear in `C:\Temp\ACESetup12.0.2.0`
   - `ACESetup12.0.2.0.exe`. This file is the installation executable file.
   - `ACEtk1.cab`. This file contains the product files for the IBM App Connect Enterprise Toolkit component.
   - `ACEae1.cab`. This file contains the product files for the OpenAPI editor component.
   - `ACEws1.cab`. This file contains the product files for the WebSphere Service Registry and Repository nodes component.
   - `readmes`. This folder contains the `readme` files for all of the supported national languages.

   All of the extracted files are required to do a full installation of the product. If you want to package the installation files into a scripted deployment of IBM App Connect Enterprise on your server environment, you can omit the files for the components that you do not want. If you want to install the OpenAPI editor component, then you must also install the IBM App Connect Enterprise Toolkit component.

   If you want to install by using the command-line, complete step 2. If you want to install by using the **Install** wizard, complete step 3.
2. Optional: To install IBM App Connect Enterprise on Windows by using the command-line options, run the following command with the options that you require:

   ```
   ACESetup12.0.n.0.exe
   ```

   For example, if you do not want to install the IBM App Connect Enterprise Toolkit component, run the command with the following options:

   ```
   ACESetup12.0.n.0.exe InstallToolkit=0
   ```

   For more information about command-line options you can use, for example, to initiate an installation in silent mode, see “Command-line options for installing IBM App Connect Enterprise on Windows” on page 78.
3. Optional: To install IBM App Connect Enterprise on Windows by using the **Install** wizard, by completing the following steps:
   a) Required: In the directory where you unpacked the installation files, right -click on the file `ACESetup12.0.n.0Setup.exe` to open the menu, and then select **Run as administrator** to

open the **Install** wizard. Alternatively, in the command prompt, navigate to the directory where you unpacked the installation files and run the following command to open the **Install** wizard:

```
ACESetup12.0.n.0.exe
```

The **Install** wizard opens in which you select the components that you want to install.

b) Click **Options** to display the components that you can install and the path for the install location.

c) Optional: By default, both the IBM App Connect Enterprise Toolkit and the IBM App Connect Enterprise runtime component are installed. If you do not want to install the IBM App Connect Enterprise Toolkit, deselect the **Install IBM App Connect Enterprise Toolkit** checkbox.

d) Optional: By default, the OpenAPI editor is installed. If you do not want to install the OpenAPI editor, deselect the **Install IBM App Connect Enterprise Open API editor** checkbox. If you want to install the OpenAPI editor component, then you must also install the IBM App Connect Enterprise Toolkit component.

e) Optional: By default, the WebSphere Service Registry and Repository nodes are not installed. If you want to install the WebSphere Service Registry and Repository nodes, select the **Install IBM App Connect Enterprise WebSphere Service Registry and Repository message flow nodes** checkbox.

If you choose not to install the WebSphere Service Registry and Repository nodes, the `<install_dir>/server/wsrrcomponent directory is not created during the installation. However, the` WebSphere Service Registry and Repository nodes still appear in the IBM App Connect Enterprise Toolkit.

f) Optional: By default IBM App Connect Enterprise is installed in `C:\Program Files\IBM\ACE \12.0.n.0`. If you want to change the installation directory, type or navigate to the directory path that you want to use.

If you specify your own directory path, be aware of the following restrictions:

- Specify a directory path to a new or empty directory. Do not install IBM App Connect Enterprise 12.0 over an existing deployment of IBM App Connect Enterprise. To upgrade the product, you can uninstall the existing version of IBM App Connect Enterprise before you install the new version of the product. Alternatively, you can install the new version alongside the existing version and migrate your resources to the new version.
- The Windows operating system has a file system limit of 256 characters for the combination of a directory path and file name. If the combination of path and resource name (for example, a message flow) exceeds this length, you might have access problems. Choose a short name for your directory path to avoid problems that are associated with this restriction.

g) Accept the license terms and conditions and then click **Install** to start the installation.

If the Microsoft.NET framework is not already installed on your computer, then the framework is installed as part of the IBM App Connect Enterprise installation. The installation of the Microsoft.NET framework might involve communicating with Microsoft servers over the internet. Depending on the configuration of your firewall, you might need to accept a prompt to allow this communication.

Results

The installation wizard proceeds to install the IBM App Connect Enterprise software. After some minutes, the installation process finishes with the message "Installation successfully completed". You can then close the installation wizard.

What to do next

If you install the IBM App Connect Enterprise Toolkit, you can verify your installation by starting the toolkit by using either of the following methods:

- Use the Windows **Start** menu option for **IBM App Connect Enterprise 12.0.*n*.0** > **12.0.*n*.0**.
- From the IBM App Connect Enterprise Console, type `ace toolkit`.

Command-line options for installing IBM App Connect Enterprise on Windows

You can use command-line options to customize the installation process.

You can use one or more of the following options with the ACESetup12.0.*n*.0.exe command:

Purpose	Command-line option
Accept the license. This option is mandatory when you run the installation in silent or passive mode.	LICENSE_ACCEPTED=true
Run the installation in silent mode; the installation is completed in the background. **Note:** You must run the command from an elevated command prompt (a command prompt where you have administrative privileges), otherwise you are prompted for an administration password.	/quiet, /q, /silent, or /s For example: `ACESetup12.0.n.0.exe /quiet LICENSE_ACCEPTED=true`
Run the installation in passive mode; installation progress is displayed but no user interaction is required. **Note:** You must run the command from an elevated command prompt (a command prompt where you have administrative privileges), otherwise you are prompted for an administration password.	/passive For example: `ACESetup12.0.n.0.exe /passive LICENSE_ACCEPTED=true`
Install IBM App Connect Enterprise to a directory of your choice.	InstallFolder="*install_location*", where *install_location* is the fully qualified path of the directory where you want to install IBM App Connect Enterprise. For example: `ACESetup12.0.n.0.exe InstallFolder="C:\ACEV12"`
Save the installation log file in a custom location. By default, the installation log is saved in the system temporary directory.	/log "*log_location*", where *log_location* is the fully qualified path to the file where you want to save the log information for the IBM App Connect Enterprise installation. For example: `ACESetup12.0.n.0.exe /log "C:\InstallLogs\ace_v12.log"`
Do not install the IBM App Connect Enterprise Toolkit.	InstallToolkit=0 For example: `ACESetup12.0.n.0.exe InstallToolkit=0`
'Do not install the OpenAPI editor.	`ACESetup12.0.n.0.exe InstallAPICeditor=0`
Install the WebSphere Service Registry and Repository nodes.	InstallWSSRnodes=1 For example: `ACESetup12.0.n.0.exe InstallWSRRnodes=1`

Purpose	Command-line option
Change the language that is used by the installation wizard. **Note:** The language of the IBM App Connect Enterprise installation wizard defaults to the language set in **Control Panel** > **Region and Language** > **Format**.	`/lang localeID`, where *localeID* is the locale ID for the language that you want the installation wizard to use. Table 10 on page 79 lists the options that you can use for the locale ID. For example, to display the installation wizard in English when your operating system locale is not set to English, use the command: `ACESetup12.0.n.0 /lang 1033`
Uninstall IBM App Connect Enterprise.	`/uninstall`. For example: `ACESetup12.0.n.0.exe /uninstall`

List the most commonly used command-line options.	`/?` or `/help`
	
	For example:
	
	`ACESetup12.0.n.0.exe /?`

Table 10. Locale ID table

Locale ID	Language code	Language
1028	zh_TW	Chinese (Traditional)
1031	de_DE	German
1033	en_US	English
1036	fr_FR	French
1040	it_IT	Italian
1041	ja_JP	Japanese
1042	ko_KR	Korean
1045	pl_PL	Polish
1046	pt_BR	Portuguese (Brazil)
1049	ru_RU	Russian
1055	tr_TR	Turkish
2052	zh_CN	Chinese (Simplified)
3082	es_ES	Spanish

Installing IBM App Connect Enterprise on Linux

You can install IBM App Connect Enterprise on Linux, either for use by a single user or for use by multiple users.

Before you begin

- Check the `readme.html` file for any updates to these installation instructions; see the product readme web page.

- Check that you have enough memory and disk space; see IBM App Connect Enterprise system requirements.
- Check that you completed any prerequisite steps; see "Preparing the system" on page 71.

About this task

As a user without administrative privileges, you can create a single-user installation of IBM App Connect Enterprise in your home directory. This single-user installation is then accessible only by your user ID.

As a user with administrative privileges, you can create a shared installation of IBM App Connect Enterprise. To authorize any users of the computer to access the shared installation of IBM App Connect Enterprise, add the users to the mqbrkrs group by using the security facilities that are provided by your operating system.

Installing the software

Procedure

To install IBM App Connect Enterprise, complete the following steps:

1. Log in to the system where you are installing IBM App Connect Enterprise.
   - If you are deploying a single-user installation, log in with your personal user ID.
   - If you are deploying a shared installation, either:
     - Log in as root or as a super user who has write access to the `/var` directory, or
     - Log in as a non-root user who is a member of mqbrkrs and has write access to the `/var/mqsi` directory.
2. Unpack the installation image by completing the following steps:
   a) Create or navigate to a directory where you have write access.
   For example, `$HOME` for a single-user installation, or `/opt/IBM` for a shared installation.
   b) Run the following command to unpack the installation image:

   ```
   tar -xzvf ace-12.0.n.0.tar.gz
   ```

   Note: On all Linux systems, except for Linux on Z, and Linux on POWER®, the installation includes the IBM App Connect Enterprise Toolkit and the WebSphere Service Registry and Repository nodes. If you do not want to install the IBM App Connect Enterprise Toolkit, you can run the following command instead:

   ```
   tar -xzvf ace-12.0.n.0.tar.gz --exclude ace-12.0.n.0/tools
   ```

   If you do not want to install the WebSphere Service Registry and Repository nodes, you can run the following command instead:

   ```
   tar -xzvf ace-12.0.n.0.tar.gz --exclude ace-12.0.n.0/server/wsrrcomponent
   ```

   If you do not want to install the IBM App Connect Enterprise Toolkit or the WebSphere Service Registry and Repository nodes, you can run the following command instead:

   ```
   tar -xzvf ace-12.0.n.0.tar.gz --exclude ace-12.0.n.0/server/wsrrcomponent
   --exclude ace-12.0.n.0/tools
   ```

   If you choose not to install the WebSphere Service Registry and Repository nodes, the `<install_dir>\server\wsrrcomponent directory is not created during the installation. However, the` WebSphere Service Registry and Repository nodes still appear in the IBM App Connect Enterprise Toolkit.

3. Accept the IBM App Connect Enterprise license by completing the following steps:
   a) Navigate to the installation directory.
      For example, *install_dir*/ace-12.0.*n*.0 where *install_dir* is the directory where you unpacked the installation image.
   b) Type one of the following commands:
      - For a single-user installation:
        - `./ace accept license`. If you use this command, you are prompted to accept the license.
        - `./ace accept license silently`. If you use this command, the license is accepted even though the license dialog is not displayed.

        The following directory is created: `$HOME/aceconfig`. This directory is the work path for IBM App Connect Enterprise, and stores the IBM App Connect Enterprise configuration files.
      - For a shared installation:
        - `./ace make registry global accept license`. If you use this command, you are prompted to accept the license.
        - `./ace make registry global accept license silently`. If you use this command, the license dialog is suppressed and the license is automatically accepted.

        The following directory is created: `/var/mqsi`. This directory is the work path for IBM App Connect Enterprise, and stores the IBM App Connect Enterprise configuration files.
4. Optional: If you installed a shared installation of IBM App Connect Enterprise, grant users access to the installation by adding them to the mqbrkrs user group.

   Note: On Linux, the user ID that installed IBM App Connect Enterprise is not automatically added to the mqbrkrs group. If you want to use this user ID with IBM App Connect Enterprise, you must add the user ID to the mqbrkrs group.

Installing IBM App Connect Enterprise on AIX

You can install IBM App Connect Enterprise on AIX, for use by a single user or multiple users.

Before you begin

- Check the `readme.html` file for any updates to these installation instructions; see the product readme web page.
- Check that you have enough memory and disk space; see IBM App Connect Enterprise system requirements.
- Check that you have completed any prerequisite steps; see “Preparing the system” on page 71.

About this task

As a user without administrative rights, you can create a single-user installation of IBM App Connect Enterprise in your home directory. This single-user installation is then accessible only by your user ID.

As a user with administrative rights, you can create a shared installation of IBM App Connect Enterprise. To authorize any users of the computer to access the shared installation of IBM App Connect Enterprise, add the users to the mqbrkrs group by using the security facilities that are provided by your operating system.

Installing the software

Procedure

To install IBM App Connect Enterprise, complete the following steps:

1. Log in to the system where you are installing IBM App Connect Enterprise.
   - If you are deploying a single-user installation, log in with your personal user ID.
   - If you are deploying a shared installation, either:
     - Log in as root or as a super user who has write access to the `/var` directory, or
     - Log in as a non-root user who is a member of mqbrkrs and has write access to the `/var/mqsi` directory.
2. Unpack the installation image by completing the following steps:
   a) Create or navigate to a directory where you have write access.
   For example, `$HOME` for a single-user installation, or `/opt/IBM` for a shared installation.
   b) Run the following command to unpack the installation image:

   ```
   zcat ace-12.0.n.0.tar.Z | tar -xvf -
   ```

3. Accept the IBM App Connect Enterprise license by completing the following steps:
   a) Navigate to the installation directory.
   For example, *`install_dir`*`/ace-12.0.`*`n`*`.0` where *install_dir* is the directory where you unpacked the installation image.
   b) Type one of the following commands:
   - For a single-user installation:
     - `./ace accept license`. If you use this command, you are prompted to accept the license.
     - `./ace accept license silently`. If you use this command, the license is accepted even though the license dialog is not displayed.

     The following directory is created: `$HOME/iibconfig`. This directory is the work path for IBM App Connect Enterprise, and stores the IBM App Connect Enterprise configuration files.

     Note: After you accept the license, IBM App Connect Enterprise does not function correctly if you copy the program code to another location. To relocate the IBM App Connect Enterprise program code, choose one of the following options:
     - To refer to IBM App Connect Enterprise as if it is installed in another location, leave the IBM App Connect Enterprise program where it is currently installed and create a symbolic link to the program. See your operating system instructions for information about creating symbolic links.
     - To move IBM App Connect Enterprise to another physical location, uninstall the IBM App Connect Enterprise program and reinstall the program in the required location.
   - For a shared installation:
     - `./ace make registry global accept license`. If you use this command, you are prompted to accept the license.
     - `./ace make registry global accept license silently`. If you use this command, the license is accepted even though the license dialog is not displayed.

     The following directory is created: `/var/mqsi`. This directory is the work path for IBM App Connect Enterprise, and stores the IBM App Connect Enterprise configuration files.
4. Optional: If you installed a shared installation of IBM App Connect Enterprise, grant users access to the installation by adding them to the mqbrkrs user group.

   On AIX, the user ID that installed IBM App Connect Enterprise is not automatically added to the mqbrkrs group. You must add this user ID to the mqbrkrs group if you want to use the user ID with IBM App Connect Enterprise.

Setting up a command environment

Runtime components and commands that administer runtime components must be run from a command environment. You must initialize this environment by running the **mqsiprofile** command.

About this task

The **mqsiprofile** command places the Version 12.0 commands and libraries at the front of your search path, and can override any combination of PATH, CLASSPATH, or library PATH.

ODBC settings on Linux systems are found in a text file that is defined by the ODBCINI environment variable. If you are using Linux, set ODBCINI to point to a copy of the sample file *install_dir*/server/ODBC/unixodbc/odbc.ini, where *install_dir* is the IBM App Connect Enterprise installation directory.

On Linux, if you want to use IBM MQ features, you must set the IBM MQ environment where you want the integration server to run; for more information, see "Setting the IBM MQ environment on Linux and AIX" on page 84.

Ensure that you use this environment each time you run an administrative command, or start an integration server.

The **mqsiprofile** command also runs extra scripts that are copied to the common\profiles directory (on Windows) or the common/profiles directory (on Linux systems). These scripts can be used to initialize the environment for runtime components and other resources, such as databases.

The following steps explain how to initialize your command environment, either by explicitly running the **mqsiprofile** command (on Linux) or by starting the command console (on Windows).

Procedure

- On Windows, open a command console by searching for **IBM App Connect Enterprise Console**. If you have multiple installations of IBM App Connect Enterprise, make sure that you are running the IBM App Connect Enterprise Console from the build of the IBM App Connect Enterprise installation that you want to administer.
- On Linux systems, locate and run the mqsiprofile.sh script in the directory in which you installed the appropriate product.

  ```
  . install_dir/server/bin/mqsiprofile
  ```

  You must include the period and space for this command to work correctly. If you want to run this command at the start of every session, add it to your login profile.

  If you use the zsh shell, running the **mqsiprofile** might cause the terminal session to exit. To resolve this issue, run the **unsetopt function_argzero** command before you run the **mqsiprofile** command.

What to do next

From the command line in the initialized environment, you can run commands to administer IBM App Connect Enterprise. For example, you can configure and start an integration server, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172 and "Starting an integration server" on page 250.

Running database setup scripts before starting an integration node

Configure an integration server to access databases from deployed message flows by customizing the command environment. This process is relevant only when you are starting an integration node.

When you install a database product, the relevant settings might be made to the system environment (typical for Windows systems). However, some database managers provide a profile to perform this setup,

or provide details of actions that you must take. Always check the database product documentation for environment setup details; the information that is provided here is for general guidance only.

If a profile is provided for the database that you are using, complete the following process:

If you can update the profile to provide permanent values for the details that are required (for example, the database server name or the installation directory), complete Step "1" on page 84. You must also complete either Step "2" on page 84 or Step "3" on page 84. If you cannot update the profile to provide permanent values, then you must complete Step "4" on page 84

1. Complete the changes to the database profile.
2. If your platform is Windows copy the profile file to the appropriate directory:
   - For an HA integration node, use *shared_work_path**config**node_name**profiles*.
   - If the integration node was created with a specified work path, which has a subdirectory *workPath**config**node_name**profiles*, then use that profiles directory.
   - In the default case, use *%MQSI_REGISTRY%**config**node_name**profiles*.
3. If your platform is Linux or UNIX copy the profile file to the appropriate directory:
   - For an HA integration node, use *shared_work_path/common/profiles*.
   - If the integration node was created with a specified work path, under which has a subdirectory *work_path/config/node_name/profiles*, then use that profiles directory.
   - In the default case, use *${MQSI_REGISTRY}/config/node_name/profiles*.
4. If you cannot update the profile permanently, but need to make changes repeatedly, you must edit it in the appropriate directory as specified in Step "2" on page 84 or Step "3" on page 84. You must edit the profile each time before you start the integration node.

Note:

The directory, *workPath* is the machine-wide IBM App Connect Enterprise working directory. To verify the machine-wide IBM App Connect Enterprise working directory, enter the following command in a command console:

```
echo %MQSI_WORKPATH%
```

The directory, *shared_work_path* is the working directory for a multi-instance integration node.

For more information about the parameters for the working directory of an integration node, see command, and command - systems.

Setting the IBM MQ environment on Linux and AIX

On Linux and AIX, configure an integration server to use the specified IBM MQ environment.

In addition to the scripts that you must run to create your default system queues (as described in "Creating the default system queues on an IBM MQ queue manager" on page 99), on Linux and AIX systems, you must configure the IBM MQ environment where you want the integration server to run, before you start it. If you do not set the environment, your integration server might not run in the expected location.

Set the environment by using the IBM MQ **setmqenv** command; for more information see the setmqenv command in the IBM MQ product documentation. For example, if you want the integration server to use the IBM MQ installation path `/opt/mqm`:

```
. /opt/mqm/setmqenv -s -x 64
```

Because the **setmqenv** command is not persistent, you can create an IBM App Connect Enterprise common profile to run this command in either of the following ways:

- Run **setmqenv** when **mqsiprofile** is run.
- Run **setmqenv** when the integration server is started.

Configuring your IBM App Connect Enterprise installation to conform to your license

You must ensure that your installation conforms to the terms of your license.

Before you begin

For a full description of the operation mode, see “Operation modes” on page 3.

Procedure

- **Upgrading from IBM App Connect Enterprise for Developers (Developer Edition)**

  If you installed the Developer Edition, and you have now purchased a fully supported version of the product such as the Advanced Edition, you can keep the components and all the associated resources that you have already created and configured. Before you install the purchased product, uninstall the Developer Edition. You can retain your existing integration servers and workspace, and any Developer mode resources remain in Developer mode by default. You can leave them in Developer mode, or you can use the **mqsimode** command to change the operation mode to reflect the license that you have purchased.

  To change the operation mode, complete the following steps.

  – If you intend to keep the integration servers that you created with the Developer Edition for further development and unit test by your developers, change the operation mode to `advanced`, as described in “Changing the operation mode” on page 86.

    Check the license agreement file to ensure that your configuration conforms to any restrictions for development and unit test. Development and unit test conditions are described in “License requirements” on page 5.

  – If you intend to use the integration nodes that you created with Developer Edition for production purposes, change the operation mode to conform to the license that you have purchased:

    - If you have purchased Standard Edition, change the operation mode to `standard`.
    - If you have purchased the Nonproduction license, change the operation mode to `nonproduction`.
    - If you have purchased the full IBM App Connect Enterprise license, change the operation mode to `advanced`.

  If you reinstall IBM App Connect Enterprise with the new package for your purchased product, the default operation mode is `advanced`. If you have purchased a full IBM App Connect Enterprise license, you can leave the operation mode as `advanced`, otherwise you must change the mode of your integration nodes to conform to the license that you have purchased (for example, `standard` mode).

- **Standard Edition**

  If you have purchased IBM App Connect Enterprise Standard Edition, and installed components from the full runtime package, read the following information.

  – All integration servers that you have created have an operation mode set to `advanced`, which is the default setting for this installation. You can keep them for further development and unit test, subject to any restrictions that apply for unit test environments, as indicated in your license. Development and unit test conditions are described in “License requirements” on page 5.

  – If you intend to use the integration servers for production purposes, follow the instructions in “Changing the operation mode” on page 86, to conform to the license that you have purchased.

    - If you have purchased Standard Edition, change the operation mode to `standard`.

- **Full (unrestricted) license for IBM App Connect Enterprise**

  If you have purchased a full (unrestricted) license and installed components from the full runtime package, all components that you create have an operation mode set to the default value of `advanced`, which is the correct setting for your license.

  You can continue to work with all the components and associated resources that you have already created. For example, after you have completed verification, you might keep the resources that you have created for further development and unit test (subject to any restrictions that apply for unit test environments, as indicated in your license). Development and unit test conditions are described in "License requirements" on page 5.

Changing the operation mode

Change the operation mode in which your integration servers are working by using the **mqsimode** command.

About this task

You must change your configuration to ensure that your integration servers are running in the operation mode for which you have purchased a license. You can change the mode to `standard`, `nonproduction`, `nonproductionfree`, or `advanced`.

Procedure

1. Run the **mqsimode** command with the **-o** parameter to change the operation mode, or without the **-o** parameter to view the current setting; for more information, see command.
2. Check for error messages. If you attempt to reconfigure your installation to a mode that is not sufficient for the deployed resources, the **mqsimode** command issues a warning indicating that changing the mode is not allowed. Resolve any violations, if required.
3. (Optional) Run the **mqsimode** command again to confirm that there are no violations.

Moving from Developer Edition

If you want to move from Developer Edition to an alternative edition, you must uninstall Developer Edition before installing the new edition.

Before you begin

Contact your IBM representative to upgrade your license.

Procedure

To move from Developer Edition to an alternative edition, complete the following steps.

1. Stop all running integration servers, as described in "Starting an integration server" on page 250.
2. Uninstall the Developer Edition.
3. Install your new licensed edition.
4. Restart your integration servers, which will now run in an unrestricted mode.

Checking the operation mode

Run the **mqsimode** command to report the operation mode that is being used by your IBM App Connect Enterprise installation.

About this task

Use the **mqsimode** command on its own, without the **-o** parameter, to receive a report about the operation mode that is being used by your installation of IBM App Connect Enterprise, and a report about any mode violations. For example:

```
mqsimode
```

Configuring the IBM App Connect Enterprise web user interface

You can use the `node.conf.yaml` and `server.conf.yaml` configuration files to configure the port that is used to connect to the web user interface, and to secure the connection.

Before you begin

Configure an integration node or server, by following the instructions described in "Configuring an integration node by modifying the node.conf.yaml file" on page 118 or "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

About this task

The IBM App Connect Enterprise web user interface enables you to access integration node or integration server resources by using a web browser, and it provides integration administrators with a method of administering those resources. For more information about the web user interface, see web user interface. To learn some basics about administering IBM App Connect Enterprise with the web user interface, see the tutorial "Getting started - Exploring the Web UI" in the IBM App Connect Enterprise Toolkit.

Procedure

1. Open the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

Set the **RestAdminListener** properties, which control the settings for the web user interface:

2. Set the **port** property to the port to be used by the web user interface and the IBM App Connect Enterprise Toolkit. By default, this port is set to 4414.

3. Optional: If you want to secure the connection, set the following properties:
   a) Set `host: 'hostname'`

      Specify the hostname where the integration node or integration server is running.

   b) If you want to use basic authentication for users who log in to the web user interface, uncomment the following property:

      basicAuth: true
      Specify whether clients require a web username and password (`true` or `false`).

      You also need to create at least one username and password, by running the **mqsiwebuseradmin** command as described in the later step section "Additional steps for security configuration".

   c) If you want to use SSL or TLS to secure the connection:

      Set the following properties:

      sslCertificate
      Specify the path to a certificate store for use in securing the REST Administration port, which is used by the Web User Interface and IBM App Connect Enterprise Toolkit, in the form `/path/to/serverPKCS.p12`.

      If you are using a `.pem` certificate, the **sslCertificate** is the full path to the server certificate key.

      If you are using `.p12` or `.pfx` certificate, the **sslCertificate** is the full path to the server certificate store file.

      The certificate store cannot be in JKS format, and hence must be a separate certificate store from the certificate store that is used by the integration server for things such as HTTPS message flow nodes.

      sslPassword
      Specify the server certificate password alias, in the form `adminRestApi::sslpwd`.

      If you are using a `.pem` certificate, the **sslPassword** is the full path to the server private key, which must be a standard private key and not an encrypted one.

      If you are using `.p12` or `.pfx` certificate, the **sslPassword** is the passphrase or alias to the passphrase of the certificate store.

      You also need to set the password to be used for your server certificate, by running the **mqsisetdbparms** command as described in the later step section "Additional steps for security configuration".

   d) If you want to use SSL client certificates (mutual authentication):

      requireClientCert
      Specify whether a certificate is to be requested from the client (true or false).

      caPath
      Specify the file path that contains certificate authority certificates; all files in this path are read.

4. If you want to view message flow statistics in the web user interface, you must also enable the reporting of message flow statistics and accounting data, as described in “Configuring the collection of message flow statistics by using a .yaml configuration file” on page 2729. The web user interface consumes message flow statistics and accounting data in JSON format, which means that you must include `json` as one of the values in the **outputFormat** property in the `.conf.yaml` configuration file.
5. If you want to view resource statistics in the web user interface, you must enable the reporting of these statistics as described in “Managing resource statistics collection” on page 2741.
6. Save the .yaml file. The properties that you set in the .yaml file take effect when the integration node or server is started. If you modify these properties again, you must also restart the integration node or server.
7. Restart the integration node or server for the changes to take effect.

Additional steps for security configuration:

8. If you set `basicAuth: true`, create one or more usernames and passwords that users must specify when they start the web user interface.

   In the IBM App Connect Enterprise command console, create a username and password by issuing the **mqsiwebuseradmin** command. For example:

   ```
   mqsiwebuseradmin -w c:\workdir\ACEServ1 -u admin -a password -c
   ```

   For more information about configuring basic authentication, see "Configuring HTTP basic authentication for an integration node or server" on page 2548.
9. If you set `sslCertificate` and `sslPassword` to use SSL or TLS to secure the connection.

   In the IBM App Connect Enterprise command console, use the **mqsisetdbparms** command to set the password to be used for your server certificate. For example:

   ```
   mqsisetdbparms -w c:\workdir\ACEServ1 -n adminRestApi::sslpwd -u dummy -p password
   ```

   For more information about configuring use of SSL or TLS to secure the connection, see "Configuring SSL or TLS for an integration node or server" on page 2548.

Results

You can now access the web user interface by opening a browser and specifying the host and port that you configured in the `.yaml` configuration file. For more information, see "Accessing the web user interface" on page 89.

Accessing the web user interface

You can use the IBM App Connect Enterprise web user interface to access integration node and integration server resources through a web browser.

Before you begin

Configure the web user interface, by completing the steps in "Configuring the IBM App Connect Enterprise web user interface" on page 87.

About this task

When the web user interface has been configured (by modifying the `node.conf.yaml` or `server.conf.yaml` configuration file), you can access it by completing the following steps:

Procedure

1. Start the web user interface, by using one of the following methods:
   - **From the IBM App Connect Enterprise Toolkit:** In the toolkit, right-click the integration node or server and select **Start web user interface** from the menu.
   - **From a web browser:** Enter the URL for the web user interface into a web browser, in the following form:

     protocol`://`*hostname*`:`*port*

     where:

     – *protocol* is either `http` or `https`
     – *hostname* is the host that you specified in the **`host`** property in the `.yaml` configuration file for the integration node or server (for example, `localhost` or `127.0.0.1`)
     – *port* identifies the port that is to be used for the web user interface and the toolkit, and must match the value that you set for the **`port`** property in the `.yaml` configuration file; by default this is set to 4414.

     For example:

     ```
     https://localhost:4414
     ```

     or

     ```
     https://127.0.0.1:4414
     ```

   If the integration server is configured to require authentication for administrative tasks (that is, if the **`basicAuth`** property in the `.yaml` configuration file is set to `true`), you are prompted to enter your user ID and password.
2. If prompted, enter your user ID and password and log in to the system.

   If the connection to the web user interface is not secured, you can access all data and resources without logging in. Users have permissions granted to them according to their role, so administrators and web users can have different access to resources based on their role. For more information about roles, see “Role-based security” on page 2503. For information about creating user accounts, see “Managing web user accounts” on page 2510.

Results

A web browser window opens, in which you can view and administer your resources.

Applying service to IBM App Connect Enterprise

Install IBM App Connect Enterprise fix packs to get the latest service updates and fixes.

About this task

Service updates and other fixes are delivered occasionally in the form of fix packs. You can find information about available fix packs on the IBM App Connect Enterprise support web page.

IBM App Connect Enterprise fix packs are complete App Connect Enterprise installations and are installed in the same way as the initial product release. For more information, see “Installing IBM App Connect Enterprise software” on page 75. You can install a fix pack on the same computer as a previous release, or on a different computer. For more information about installing more than one fix pack (at the same or different levels) on a single computer, see “Coexistence of IBM App Connect Enterprise 12.0 with previous versions” on page 70.

When you have installed a fix pack, see New function added in and subsequent modification and fix packs for information about the function that is provided in that fix pack and in any previous fix packs.

IBM App Connect Enterprise supports interoperability between different fix packs of the same version. However, errors might occur if you create (or edit) an artifact or enable a new feature, and then use the artifact with a component that is using an earlier fix pack. For example, the following situations might generate errors:

- Editing source artifacts in the IBM App Connect Enterprise Toolkit that were created on an IBM App Connect Enterprise Toolkit that is using a later fix pack.
- Building BAR files from artifacts that were created on an IBM App Connect Enterprise Toolkit that is using a later fix pack than the build system.
- Deploying BAR files that were built by using a component that is using a later fix pack than the target integration server.
- Connecting a REST administration application (including the IBM App Connect Enterprise Toolkit and commands) to a remote integration server that is using a later fix pack.

It is good practice to ensure that downstream components (such as integration servers and build systems) are using the same or later fix pack as upstream components (such as the IBM App Connect Enterprise Toolkit). An alternative option is to ensure that you install the earliest common fix pack on the build systems so that you are warned about any incompatible options as early as possible.

If you installed the fix pack on the same computer as an existing installation of IBM App Connect Enterprise 12.0, complete the following steps to convert your integration nodes to run with the new code. These methods are valid only for switching an integration node between different fix pack levels that are associated with the same product code version. You must use the **mqsiextractcomponents** command to migrate an integration node to a different version and release.

Procedure

- Windows

  On Windows, complete one of the following steps:

  - Start the IBM App Connect Enterprise Console for the new fix pack level, and then start the integration node by using the **mqsistart** command.
  - Start the IBM App Connect Enterprise Toolkit for the new fix pack level. Any local integration nodes that are stopped and configured to start automatically with the IBM App Connect Enterprise Toolkit use the product code that is associated with the version of the IBM App Connect Enterprise Toolkit.
  - Start the integration node by using a script that first runs the **mqsiprofile** for the new fix pack level.
  - If your integration node is configured to start as an IBM MQ service, you must manually update the IBM MQ Service definition on the queue manager that is associated with the integration node. This update enables the integration node to use the new code. For more information, see “Applying service to an integration node that starts under the control of an IBM MQ queue manager” on page 92.

- Linux

  On Linux:

  - Start the IBM App Connect Enterprise Toolkit for the new fix pack level. Any local integration nodes that are stopped and configured to start automatically with the IBM App Connect Enterprise Toolkit use the product code that is associated with the version of the IBM App Connect Enterprise Toolkit.
  - Run the **mqsiprofile** command from the new fix pack level, and start your integration node by using the **mqsistart** command.
  - If your integration node is configured to start as an IBM MQ service, you must manually update the IBM MQ Service definition on the queue manager that is associated with the integration node. This update enables the integration node to use the new code. For more information, see “Applying service to an integration node that starts under the control of an IBM MQ queue manager” on page 92.

- AIX
  On AIX :
  - Run the **mqsiprofile** command from the new fix pack level, and start your integration node by using the **mqsistart** command.
  - If your integration node is configured to start as an IBM MQ service, you must manually update the IBM MQ Service definition on the queue manager that is associated with the integration node. This update enables the integration node to use the new code. For more information, see "Applying service to an integration node that starts under the control of an IBM MQ queue manager" on page 92.

What to do next

You can use any Version 11.0 fix pack level of the IBM App Connect Enterprise Toolkit to deploy integration solutions to an integration server that runs any Version 11.0 fix pack level.

Applying service to an integration node that starts under the control of an IBM MQ queue manager

If an integration node starts under the control of an IBM MQ queue manager, you must manually update the WebSphere MQ Service definition on the queue manager that is associated with the integration node. This update enables the integration node to use the new code.

Procedure

To update the IBM MQ Service definition on the queue manager, complete the following steps:

1. At a command line, type `runmqsc qmgr` where *qmgr* is the name of the queue manager that is associated with the integration node.
2. To display the current settings for the integration node type `display service('serviceName')`, where *serviceName* is the name of your IBM MQ service.

   The output details the current settings, for example, when your integration node is running the IBM App Connect Enterprise Version 11.0 code, you might see the following output:

   ```
   AMQ8629: Display service information details.
   SERVICE(MQSI_integrationNodeName) CONTROL(QMGR) SERVTYPE(COMMAND)
   STARTCMD(C:\PROGRA~1\IBM\ACE\1100~1.0\server\bin\runMQService.bat)
   STARTARG(C:\PROGRA~1\IBM\ACE\1100~1.0\server\bin integrationNodeName)
   STOPCMD(C:\PROGRA~1\IBM\ACE\1100~1.0\server\bin\endMQService.bat)
   STOPARG(C:\PROGRA~1\IBM\ACE\1100~1.0\server\bin integrationNodeName)
   STDOUT(C:\PROGRA~3\IBM\MQSI\common\log\integrationNodeName.mqservice.log)
   STDERR(C:\PROGRA~3\IBM\MQSI\common\log\integrationNodeName.mqservice.log)
   DESCR( ) ALTDATE(2019-12-19) ALTTIME(11.07.52)
   ```

3. Provide new fully qualified values for the following parameters: **STARTCMD**, **STARTARG**, **STOPCMD**, and **STOPARG**, but preserve the existing command names.

   For example, you might type the following command (on a single line) at the **runmsc** command line to change the integration node to use the IBM App Connect Enterprise Version 11.0.0.n code:

   Note: If you are on Windows, you must use short names .

   ```
   alter service('MQSI_integrationNodeName')
   startcmd('C:\PROGRA~1\IBM\ACE\1100~1.n\server\bin\runMQService.bat')
   startarg('C:\PROGRA~1\IBM\ACE\1100~1.n\server\bin integrationNodeName')
   stopcmd('C:\PROGRA~1\IBM\ACE\1100~1.n\server\bin\endMQService.bat')
   stoparg('C:\PROGRA~1\IBM\ACE\1100~1.n\server\bin integrationNodeName')
   ```

   where `C:\PROGRA~1\IBM\ACE\1100~1.1` is the short name for `C:\Program Files\IBM\ACE\11.0.0.1`

   If the IBM MQ service definition is changed, the message `AMQ8624: IBM MQ service changed.` is displayed.

Uninstalling IBM App Connect Enterprise

Remove IBM App Connect Enterprise, IBM App Connect Enterprise service, or other IBM App Connect Enterprise features from your computer.

Procedure

1. Check that your user ID has the correct permissions to uninstall IBM App Connect Enterprise or IBM App Connect Enterprise features.

   The requirements are defined in "Security requirements" on page 72.
2. Follow the instructions that are provided in the appropriate topic:
   - "Uninstalling IBM App Connect Enterprise on Windows" on page 93
   - "Uninstalling IBM App Connect Enterprise on Linux and UNIX systems" on page 94

What to do next

You might also want to uninstall complementary products. See the documentation that is provided by those products to complete this task.

- IBM DB2: See the DB2 product documentation.
- IBM License Metric Tool: See the IBM License Metric Tool website.

Uninstalling IBM App Connect Enterprise on Windows

You can uninstall IBM App Connect Enterprise on Windows by using the Windows Control Panel, or by using silent mode.

Before you begin

Delete any integration servers that are not required by removing the contents of their work directories.

Procedure

Uninstall IBM App Connect Enterprise on Windows by completing the following steps.

1. Log on to the computer on which IBM App Connect Enterprise is installed.
2. Uninstall IBM App Connect Enterprise by using one of the following methods:
   - Windows Control Panel: Select **Control Panel** > **Programs** > **Programs and Features**, select the IBM App Connect Enterprise deployment that you want to uninstall, and click **Uninstall**.
   - Silent mode: Navigate to the directory where the installation file is located, and type the following command from an elevated command prompt (a command prompt where you have administrative privileges):

     ```
     ACESetup12.0.n.0.exe /q /uninstall
     ```

Results

You uninstalled IBM App Connect Enterprise.

Note: The following entities are created after the installation of IBM App Connect Enterprise and so they are not removed by the uninstallation wizard. You can manually remove the entities if you do not want to keep them.

- The IBM App Connect Enterprise configuration information in the work path directory; by default the work path directory is `C:\ProgramData\IBM\MQSI`.
- The IBM App Connect Enterprise Toolkit workspace; by default the workspace is `C:\Users\user_name\IBM\ACET12\workspace`.

- The IBM App Connect Enterprise Toolkit configuration folder; by default the configuration folder is `C:\Users\user_name\AppData\Local\IBM\ACET12-config\12.0.n.0`.
- The Windows services that are associated with each local integration node that is defined on your computer (if you did not delete the integration nodes before you uninstalled IBM App Connect Enterprise). For example, IBM Integration Bus Component TESTNODE_*user_name*.

Uninstalling IBM App Connect Enterprise on Linux and UNIX systems

You can uninstall IBM App Connect Enterprise on Linux and UNIX systems by using the same user ID that you used to install the product.

Before you begin

Delete any integration servers that are not required, by removing the contents of their work directories.

Procedure

1. Log on with the user ID that you used to install IBM App Connect Enterprise.
2. Delete the IBM App Connect Enterprise entities that you do not want to keep:
   - The IBM App Connect Enterprise installation directory. For example, `install_dir/ace-12.0.n.0`.
   - (Linux only) The IBM App Connect Enterprise Toolkit workspace directory. For example, `$HOME/IBM/ACET12/workspace`. Do not delete the workspace if you want to keep any development resources that you created, such as applications or libraries. You can use existing applications and libraries with another installation of IBM App Connect Enterprise.
   - The IBM App Connect Enterprise work path directory. For example, `$HOME/iibconfig` for a single-user deployment or `/var/mqsi` for a shared deployment. Do not delete the work path directory if you want to keep any integration nodes and integration servers that you created. You can use existing integration nodes and integration servers with another installation of IBM App Connect Enterprise.

Results

You uninstalled IBM App Connect Enterprise.

Uninstalling language packs from the IBM App Connect Enterprise Toolkit

You can remove language packs from the IBM App Connect Enterprise Toolkit.

About this task

If you have installed one or more language packs for the IBM App Connect Enterprise Toolkit, you can remove any of these language packs by using one of the following methods:

- "Uninstalling language packs by using the IBM App Connect Enterprise Toolkit" on page 94
- "Uninstalling language packs by using the command line" on page 95

Uninstalling language packs by using the IBM App Connect Enterprise Toolkit

Procedure

1. Start the IBM App Connect Enterprise Toolkit.
2. From the menu, click **Help** > **About IBM App Connect Enterprise Toolkit**.

   The **About IBM App Connect Enterprise Toolkit** dialog is displayed.
3. Click **Installation Details**.

   The **IBM App Connect Enterprise Toolkit Installation Details** dialog is displayed and all the language packs that are installed are listed.

4. Select the language packs that you want to remove, click **Uninstall**, and then click **Finish**.

   The selected language packs are uninstalled and you are prompted to restart the IBM App Connect Enterprise Toolkit.
5. Click **Yes** to restart the IBM App Connect Enterprise Toolkit.

   If your operating system locale is configured for one of the languages that is supported by the language packs you uninstalled, the IBM App Connect Enterprise Toolkit user interface is now displayed in English.

Results

You have uninstalled one or more language packs from the IBM App Connect Enterprise Toolkit.

Uninstalling language packs by using the command line

Procedure

At the command line, type the following command on a single line:

- On Windows:

```
InstallPath\common\jdk\jre\bin\java
-jar InstallPath\tools\plugins\org.eclipse.equinox.launcher_1.3.0.v20130327-1446.jar
-launcher InstallPath\tools\eclipse
-application org.eclipse.equinox.p2.director
-uninstallIU LanguagePack
```

- On Linux:

```
InstallPath/common/jdk/jre/bin/java
-jar InstallPath/tools/plugins/org.eclipse.equinox.launcher_1.3.0.v20130327-1446.jar
-launcher InstallPath/tools/eclipse
-application org.eclipse.equinox.p2.director
-uninstallIU LanguagePack
```

where:

InstallPath
Specifies the directory where IBM App Connect Enterprise is installed.

LanguagePack
Specifies the language pack or language packs that you want to uninstall:

- To uninstall Language Pack 1 only, type `com.ibm.etools.ace.toolkit.nl1.feature.group`
- To uninstall Language Pack 2 only, type `com.ibm.etools.ace.toolkit.nl2.feature.group`
- To uninstall Language Pack 2a only, type `com.ibm.etools.ace.toolkit.nl2a.feature.group`
- To uninstall more than one language pack, separate the entries with a comma.

For example, to uninstall Language Pack 1 and Language Pack 2 on Windows, where IBM App Connect Enterprise is installed in the default location, type the following command on a single line:

```
"C:\Program Files\IBM\ACE\12.0.n.0\common\jdk\jre\bin\java"
-jar "C:\Program Files\IBM\ACE\12.0.n.0\tools\plugins
\org.eclipse.equinox.launcher_1.3.0.v20120522-1813.jar"
-launcher "C:\Program Files\IBM\ACE\12.0.n.0\tools\eclipse"
-application org.eclipse.equinox.p2.director
-uninstallIU
 com.ibm.etools.ace.toolkit.nl1.feature.group,com.ibm.etools.ace.toolkit.nl2.feature.group
```

A message is displayed listing the language packs that are being uninstalled. When the uninstallation is complete, a message is displayed that details how long the uninstallation took. For example:

```
Uninstalling com.ibm.etools.ace.toolkit.nl1.feature.group 11.0.0.v20180911-1900.
Uninstalling com.ibm.etools.ace.toolkit.nl2.feature.group 11.0.0.v20180911-1900.
Operation completed in 53088 ms.
```

Results

You have uninstalled one or more language packs by using the command line.

Installing and uninstalling complementary products

IBM App Connect Enterprise works with several other products to provide complementary services.

About this task

If you want to use these optional services in your IBM App Connect Enterprise environment, refer to the following installation information:

Installing IBM MQ

When you purchase a license for IBM App Connect Enterprise, your license entitles you to install IBM MQ for use by App Connect Enterprise, within the terms of the license.

About this task

IBM MQ is fully supported by IBM App Connect Enterprise, and a subset of IBM App Connect Enterprise capabilities require access to IBM MQ components.

- Some capabilities require IBM MQ to be part of the IBM App Connect Enterprise infrastructure, and you must associate a queue manager with the integration server.
- Some capabilities do not require IBM MQ to be part of the IBM App Connect Enterprise infrastructure, and you do not always need to associate a queue manager with the integration server.
- Some capabilities require access to IBM MQ, either locally or on a remote server.

Capabilities that require IBM MQ to be part of the IBM App Connect Enterprise infrastructure

For these capabilities, you must install an IBM MQ server on the same machine as the integration server that is to use the capabilities, and you must associate a queue manager with the integration server or integration node by setting the **`defaultQueueManager`** configuration property. For more information about setting the configuration property, see either “Configuring an integration server by modifying the server.conf.yaml file” on page 172 or “Configuring an integration node by modifying the node.conf.yaml file” on page 118.

For some capabilities, you must also complete additional configuration actions indicated in the "Additional configuration notes" column.

Capability	Detail	Additional configuration notes
Managed file support	You have any of the following nodes in your message flows: • node • node • node • node	
High availability	You have multi-instance integration nodes.	
High availability	You are using the HTTP proxy servlet with your integration node.	You must create the set of IBM App Connect Enterprise queues on your queue manager; see “Creating the default system queues on an IBM MQ queue manager” on page 99.

Capability	Detail	Additional configuration notes
Load balancing	You are using an integration node listener (instead of embedded listeners) with your integration node.	You must create the set of IBM App Connect Enterprise queues on your queue manager; see “Creating the default system queues on an IBM MQ queue manager” on page 99.
Global coordination of transactions (XA)	You are using globally-coordinated (XA) transactions.	

Capabilities that do not require IBM MQ to be part of the infrastructure for your IBM App Connect Enterprise deployment

For these capabilities, you must install either an IBM MQ client or an IBM MQ server on the same machine as the integration server or integration node that is to use the capabilities, but you do not need to associate a queue manager with the integration server or integration node unless specified in the "Additional configuration notes" column:

Capability	Detail	Additional configuration notes
Java connectivity and JDBC	You have one of the following nodes in your message flows: • node • node • node	

Capabilities that require access to IBM MQ, either locally or on a remote server

For these capabilities, you must install an IBM MQ client or server either on the same machine as the integration server or on another machine that can be accessed through a client connection:

Capability	Detail	Additional configuration notes
IBM MQ connectivity	You have any of the following nodes in your message flows: • node • node • node • node	You need to specify a queue manager on the integration server or integration node only if you want to use the queue manager by default for your MQ connection. For more information, see “Configuring a local connection to IBM MQ” on page 744 and “Using a remote default queue manager” on page 750.
Advanced data processing	You have any of the following nodes in your message flows: • node • node • node • node • node • node • node	You must create the set of IBM App Connect Enterprise queues on your queue manager, as described in “Creating the default system queues on an IBM MQ queue manager” on page 99. You must configure the default queue manager by setting either the **defaultQueueManager** or **remoteDefaultQueueManager** property. For more information, see “Processing events” on page 1819.

Capability	Detail	Additional configuration notes
	• node	An integration server that supports this capability must not share a queue manager with another integration server unless a unique queue prefix is set. You set a unique queue prefix either by using a policy or, if a remote default queue manager is being used, by setting the **replacementQueuePrefix** property. For more information, see "Configuring an integration server to use a remote default queue manager" on page 751. For TimeoutNotification nodes using `Controlled` mode on a managed integration server, a local default queue manager is required. For TimeoutNotification nodes using `Controlled` mode on an independent integration server, either a local default queue manager or a remote default queue manager is required. For TimeoutNotification nodes using `Automatic` mode, no queue manager is required.
Publish and subscribe	You are using publish and subscribe capabilities to emit events or accounting and statistics data over IBM MQ and MQTT. For more information, see "Configuring the publication of event messages" on page 2807.	By default, events use the queue manager that is associated with the integration server or integration node. However, you can use a policy to configure events to use a different queue manager.
Transactional support for SAP	You have any of the following nodes in your message flows: • node • node • node	You must associate a queue manager with the integration server or integration node, or use a SAPConnection policy and specify a CCDT through the **Shared TID store client definition file** property. The **remoteDefaultQueueManager** setting does not apply to these nodes, but the CCDT can specify a remote queue manager to achieve the same effect.

Procedure

1. To install IBM MQ components, see the IBM MQ product documentation: http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ins.doc/q008250_.htm.
2. For capabilities listed in Capabilities that require IBM MQ to be part of the IBM App Connect Enterprise infrastructure, associate a queue manager with the integration server or integration node that is to use the capabilities.
3. Complete any additional configuration actions for the capability that you want to use, as indicated in the "Additional configuration notes" column of the tables above.

Creating the default system queues on an IBM MQ queue manager

You can create the default system queues on the queue manager that is associated with your integration server, by running a script.

Before you begin

- You must have an IBM MQ server and a queue manager to use with your IBM App Connect Enterprise deployment. For more information about installing IBM MQ components, see the IBM MQ product documentation: https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ins.doc/q008320_.htm.
- You must be a member of the mqm and mqbrkrs operating system groups.
- Linux Ensure that the command environment is initialized, by running the **mqsiprofile** script command. For more information, see "Setting up a command environment" on page 64.

About this task

Some IBM App Connect Enterprise capabilities require access to IBM MQ components. Some of these capabilities require a set of default system queues to be created on the queue manager that is associated with the integration server. The default system queues are used to store information about in-flight messages. For more information about the capabilities that require the default system queues, see "Installing IBM MQ" on page 96.

Note: By default, running the **iib_createqueues** command disables IBM MQ channel authentication for the specified queue manager. If you want to enable channel authentication, you can modify the sample MQSC script, **iib_queues_create.mqsc**, that the command **iib_createqueues** uses to define queues.

You cannot use a secured queue manager as the local default queue manager for an integration node or an integration server.

Complete the following steps to create the default system queues on Windows and Linux.

Procedure

1. From the command environment (on Linux) or the IBM App Connect Enterprise Console (on Windows), navigate to the following directory:
   - Windows *install_dir*\server\sample\wmq
   - Linux *install_dir*/server/sample/wmq
2. Run the following command:
   - Windows iib_createqueues.cmd *qmgr_name*
   - Linux ./iib_createqueues.sh *qmgr_name iib_group*

   The *qmgr_name* parameter specifies the name of the IBM MQ queue manager where you want to create the queues.

   The *iib_group* parameter specifies the primary group of the system account that runs the integration server.

Results

The default IBM App Connect Enterprise queues are created on the queue manager.

To check that the queues were created, type the following commands:

```
runmqsc qmgr_name
```

```
display queue(SYSTEM.BROKER*)
```

A message is returned that lists the queues that were created:

```
1 : display queue(SYSTEM.BROKER*)
AMQ8409: Display Queue details.
 QUEUE(SYSTEM.BROKER.ADAPTER.FAILED)     TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.ADAPTER.INPROGRESS) TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.ADAPTER.NEW)        TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.ADAPTER.PROCESSED)  TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.ADAPTER.UNKNOWN)    TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.ADMIN.STREAM)       TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AGGR.CONTROL)       TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AGGR.REPLY)         TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AGGR.REQUEST)       TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AGGR.TIMEOUT)       TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AGGR.UNKNOWN)       TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.AUTH)               TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.CD.MODEL)           TYPE(QMODEL)
   QUEUE(SYSTEM.BROKER.CONTROL.QUEUE)      TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.DC.AUTH)            TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.DC.BACKOUT)         TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.DC.RECORD)          TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.DEFAULT.STREAM)     TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.EDA.COLLECTIONS)    TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.EDA.EVENTS)         TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.FTE.MODEL)          TYPE(QMODEL)
   QUEUE(SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS) TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.MODEL.QUEUE)        TYPE(QMODEL)
   QUEUE(SYSTEM.BROKER.SEQ.EXPIRY)         TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.SEQ.GROUP)          TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.SEQ.NUMBER)         TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.TIMEOUT.QUEUE)      TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.WS.ACK)             TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.WS.INPUT)           TYPE(QLOCAL)
   QUEUE(SYSTEM.BROKER.WS.REPLY)           TYPE(QLOCAL)
```

Note: You can delete the queues that are created for an integration server on a specified queue manager, by running the following MQSC command from the same directory:

```
runmqsc qmgr_name < iib_queues_delete.mqsc
```

For more information about the **runmqsc** command, see https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q083460_.htm.

Installing IBM License Metric Tool

IBM License Metric Tool (ILMT) enables you to monitor the use of IBM (and other) software products.

About this task

Use ITLM to perform the following software auditing functions:

- Monitor the licenses used by different machines.
- Help keep unnecessary licenses to a minimum.
- Guard against software license compliance problems.

Ensure that you choose the correct ILMT license for the IBM App Connect Enterprise edition that you have purchased. See “Operation modes” on page 3.

If you are using one or more of the WebSphere Adapters with IBM App Connect Enterprise, you must activate ILMT to include those adapters, for example in monitoring activities. If you require this support, follow the instructions in “Activating IBM License Metric Tool for WebSphere Adapters” on page 1108.

If you intend to use any of the IBM App Connect Enterprise features that require MQ functionality, you can install and use IBM MQ within the terms of your IBM App Connect Enterprise license. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

To find out more about using ILMT to monitor usage of IBM App Connect Enterprise and other IBM products, or to purchase ILMT, see the IBM License Metric Tool website.

For information about installing this product, see the IBM License Metric Tool website.

Chapter 4. Configuring IBM App Connect Enterprise

You can set up your application development environment to create, test, and deploy message flows and associated resources, and then configure IBM App Connect Enterprise to run in a production environment, either on premises or in a container. Alternatively, you can download an image from `https://hub.docker.com/r/ibmcom/ace/` and run it in a Docker container, or you can create your own Docker images.

About this task

IBM App Connect Enterprise provides an integration server component and an integration node component. You can configure an integration node as the central point of administrative control over a set of owned integration servers (typically in an on-premise environment), or you can configure integration servers that are run independently from any integration node (typically in a container environment).

This flexibility enables you to develop a solution that is best suited to the environment in which you intend to run your App Connect Enterprise applications. You can use integration nodes to provide management and control of your integration server processes, and to configure administration security for your resources. Alternatively, you can choose to develop and deploy your applications to an independent integration server quickly and easily, without the need to create and configure an integration node.

For information about configuring integration nodes and integration servers, see “Configuring integration nodes” on page 113 and “Configuring integration servers” on page 167.

Procedure

1. Set up your development environment, as described in “Configuring the IBM App Connect Enterprise Toolkit” on page 103.
2. Configure IBM App Connect Enterprise to run in a production environment, by following the instructions in one of the following topics:
   - “Configuring App Connect Enterprise to run on premises (on a physical or virtual machine)” on page 110
   - “Configuring App Connect Enterprise to run in a container” on page 111
   - “Configuring App Connect Enterprise to run in Docker” on page 111

Configuring the IBM App Connect Enterprise Toolkit

Set up your application development environment on Linux on x86-64 or Windows 64-bit to create, test, and deploy message flows and associated resources. You can configure various settings in the IBM App Connect Enterprise Toolkit to suit your requirements and your working environment.

About this task

Use the IBM App Connect Enterprise tutorials to help you get started with developing integration solutions. The tutorials are in the Tutorials Gallery, which you can find by clicking **Help** > **Tutorials Gallery** in the IBM App Connect Enterprise Toolkit. When you have tried out your first few tutorials, you can set up an environment that can support your application developers.

When you start the IBM App Connect Enterprise Toolkit, a workbench session opens, which you can use to create, configure, and manage your application development resources. You can configure your workbench session in various ways to suit your working environment and preferences. These options are described in “Configuring the IBM App Connect Enterprise Toolkit” on page 103.

If you are planning to run your IBM App Connect Enterprise integrations in a container, you will need to create and configure your integration servers. If you are planning to run your integrations on premises (on

a physical or virtual machine), you will need to create your integration servers and at least one integration node.

The following topics show you how to configure aspects of the IBM App Connect Enterprise Toolkit.

Results

A minimum display resolution of at least 1024 x 768 is required for some dialog boxes, such as the **Preferences** dialog box.

Starting the App Connect Enterprise Toolkit

Use the development environment on Linux or Windows to develop your message flows.

Before you begin

Install the IBM App Connect Enterprise Toolkit; for more information, see "Installing IBM App Connect Enterprise software" on page 75.

Procedure

To start using your development environment, complete the following steps.

1. Start the App Connect Enterprise Toolkit in the following way:

   - Linux From the installation directory, for example `$HOME/ace-11.0.0.0`, type the following text:

     ```
     ./ace toolkit
     ```

   - Windows Search for **IBM App Connect Enterprise Toolkit 12.0.n** and start it.

   Your IBM App Connect Enterprise Toolkit session opens, and displays the **Welcome** page. You can tailor various settings in the IBM App Connect Enterprise Toolkit to suit your requirements and your working environment; for example, the colors and fonts. These options are described in "Configuring the IBM App Connect Enterprise Toolkit" on page 103.

2. Close the **Welcome** page.

   You can return to the **Welcome** page later by clicking **Help** > **Welcome**. If more than one option is listed, select IBM App Connect Enterprise.

3. In the Integration Development perspective, select the Integration Explorer view.

   Your configuration might be affected by the operation mode in which IBM App Connect Enterprise is running. The default operation mode is advanced mode, in which your integration servers run with no restrictions. Check with your integration administrator which operation mode applies to your organization; you might have to change the mode of your integration node or servers after you have created them. For more information about operation modes, see "Operation modes" on page 3.

4. If you are planning to run your integration in a container, you must configure at least one integration server.

   For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

5. If you are planning to run your integration on premises (on a physical or virtual machine), you must have at least one integration node with at least one associated integration server.

   For information about how to create an integration node, see "Creating an integration node" on page 114. To add integration servers to your integration node, right-click the integration node, and click **New** > **Integration Server**. Enter a name for your integration server, and click **OK**.

   The integration server is the runtime environment in which your message flows run. You can create many integration servers on a single integration node, and you can deploy your message flows to one or more integration servers on one or more integration nodes.

   For more detailed instructions about this task, see "Creating an integration server" on page 168.

Results
Your configuration is now ready to use.

What to do next
You can start to develop resources to deploy to your integration server. You can create your own message flows, or you can use the patterns and samples that are provided to get you started. For details of these options, see Chapter 6, "Developing integration solutions," on page 481.

This task has covered the minimum set of steps that you must complete to create an integration server and (optionally) an integration node, and to configure your development environment. Typically, as an application developer, you are working in a single platform environment to create, deploy, and test your message flows before they are ready for use in a test of production environment.

More options are available for enhanced configurations, including policies. When the requirements of your message flows extend beyond this basic configuration, and you need additional configuration to support those requirements, you can find details of these more advanced options in "Configuring App Connect Enterprise to run on premises (on a physical or virtual machine)" on page 110 and "Configuring App Connect Enterprise to run in a container" on page 111.

Changing IBM App Connect Enterprise Toolkit preferences

The IBM App Connect Enterprise Toolkit has many preferences that you can change to suit your requirements. Some of these preferences are specific to IBM App Connect Enterprise. Other preferences control more general options, such as the colors and fonts in which information is displayed.

Procedure

To access the IBM App Connect Enterprise Toolkit preferences, complete the following steps.

1. Select **Window** > **Preferences**.
2. Click the plus sign that is associated with **General**, typically the first entry in the left pane.

   An expanded list of options appears, Select the aspect of the IBM App Connect Enterprise Toolkit that you want to modify. These options might be of interest:

   Startup and shutdown
   Display or suppress the dialog that prompts you to confirm the workspace location when you start IBM App Connect Enterprise Toolkit. By default, the dialog is suppressed.

   Specify whether to display the dialog box that prompts you to confirm shutdown of the IBM App Connect Enterprise Toolkit.

   Appearance
   Change the default fonts and colors that appear in the IBM App Connect Enterprise Toolkit.

   Perspectives
   Choose whether to open a new perspective in a new window.
3. When you have made your changes, click **OK** to close the Preferences dialog.

What to do next

Below the General category in the Preference dialog, there are items that refer specifically to IBM App Connect Enterprise resources, such as message flows. Review the following topics for information about setting preferences and other values that are specific to your use of these resources:

- Message flow preferences
- "Changing ESQL preferences" on page 1625
- "Message Sets: Configuring message set preferences" on page 2249
- "Testing and troubleshooting message flows" on page 647

Changing workbench capabilities

You can configure the workbench to disable access to some of the functional capabilities of IBM App Connect Enterprise.

About this task

Capabilities is an Eclipse concept that allows you to enable or disable the components of a product. By default, all components of the workbench are enabled.

To access the workbench capabilities:

Procedure

1. Select **Window** > **Preferences**.
2. Click the plus sign associated with **General**.

   An expanded list of options appears.
3. Click **Capabilities**.

   You can use the capabilities that are listed to enable or disable various product components; the capabilities are grouped according to a set of predefined categories.
4. Select **IBM Integration Toolkit** from the list of capabilities that is displayed, and select the **Advanced** button.

   A window opens that has a check box for each of the predefined categories.
5. Select the check boxes for the categories that you want to either enable or disable; click either the **Enable All** or **Disable All** button and click **OK**.

   A pane describes the functionality that is enabled following this action.

What to do next

The predefined categories for the workbench are listed together with a reference to more information about the relevant functional area of IBM App Connect Enterprise:

- **IBM Integration Toolkit - Administration**. See Chapter 5, “Administering IBM App Connect Enterprise,” on page 243.
- **IBM Integration Toolkit - Core**. See #unique_127.
- **IBM Integration Toolkit - Development**. See Chapter 6, “Developing integration solutions,” on page 481.

Configuring CVS to run with the IBM App Connect Enterprise Toolkit

Install CVS as a normal program by following the usual prompts. Not all versions of *CVSNT* are supported by Eclipse.

Procedure

1. Configure CVS by carrying out the following tasks:
   a) Create a directory on your computer, for example, on Windows - `c:\CVSRepository`.
   b) Start the *CVSNT* control panel.
      Select **Start** > **Programs** > **CVSNT** to see the icon on the desktop.
   c) Stop both the `CVS Service` and the `CVS Lock Service`.
   d) Select the `Repositories` tag, click **Add** and create a new repository.
      Note that no entry appears on the screen the first time that you do this.
   e) Use the `...` button on the next window to select the directory that you created in step "1.a" on page 107 and click **OK**.
      Note that when CVS has finished formatting its repository the backslash in the directory name is changed to a forward slash.
   f) Select the `Service Status` tab and restart both the `CVS Service` and the `CVS Lock Service`.
2. Enable the CVS Revision tag to be populated in the Eclipse Version fields in the IBM App Connect Enterprise Toolkit.
   To do this on Windows:
   a) Select `Window->Preferences`
   b) Expand the `Team` section and click **CVS**
   c) Use the drop down in the **Default keyword substitution:** field and set the value to `ASCII with keyword expansion(-kkv)`
3. Add the IBM App Connect Enterprise file types to the Eclipse CVS configuration.
   To do this:
   a) Select **File Content** in the `Team` section of the window you opened in step "2" on page 107
   b) Click **Add** and add `msgflow` as an allowable file extension.
      Ensure that the value is set to ASCII.
   c) Repeat the above procedure for the following file extensions that the integration node uses:
      - `esql`
      - `mset`
      - `mxsd`

      If you use CVS to store other file types, for example, COBOL copybooks add the appropriate file types as well.
   d) Click **OK** when you have finished.

Configuring the IBM App Connect Enterprise Toolkit to run Rational ClearCase

To use Rational® ClearCase® with the IBM App Connect Enterprise Toolkit, install Rational ClearCase Remote Client for Eclipse.

About this task

To enable Rational ClearCase in the IBM App Connect Enterprise Toolkit you must install Rational ClearCase Remote Client for Eclipse version 7.1.2.9 or later; see http://www.ibm.com/support/knowledgecenter/SSSH27_7.1.2/com.ibm.rational.clearcase.cc_ms_install.doc/topics/

t_install_ccrc_eclipse.htm. Follow the instructions to install Rational ClearCase Remote Client for Eclipse on Eclipse 3.4.

For information about the system requirements for Rational ClearCase Remote Client for Eclipse, see http://www.ibm.com/support/docview.wss?uid=swg21224586.

Results

After you enable the ClearCase capability, the ClearCase menu is displayed in the Integration Development perspective.

What to do next

To work with ClearCase:

1. Click **ClearCase** > **Connect to Rational ClearCase**.
2. Right-click your project and click **Team** > **Add to Version Control** to add your projects to the ClearCase source control.
3. After you add your projects to the ClearCase source control, you can perform ClearCase operations.

Creating a working set

Create a working set to limit the number of resources that are displayed in the Application Development view.

Before you begin

To read about working sets, see Working sets.

About this task

By creating and using a working set, you can reduce the visual complexity of what is displayed in the Application Development view, making it easier to manage and work with your projects.

To create a new working set, complete the following steps:

Procedure

1. Click the arrow on the toolbar of the Application Development view, then click **Select Working Set**.

   A list is displayed containing existing working sets. A working set is created automatically when you create an application or library. The names of these working sets are enclosed by brackets; for example [Application1].
2. To open the **New Working Set** wizard, click **New**.
3. Select a type for the new working set (for example, Integration node), then click **Next**.
4. Enter a name for the working set.
5. Select the resources that you want to include in this working set. You can also include all the projects that are dependent on your selected resources by selecting **Automatically include dependent projects in this working set**.
6. Click **Finish**.

Results

The new working set is created.

What to do next

- To show only the resources in a working set, click the arrow on the toolbar of the Application Development view, click **Select Working Set**, then select the relevant working set. If you select the

working set that is created automatically for an application or library (as indicated by brackets; for example, [Application1]), the Application Development view shows only the selected container and its contents.

- To edit the selected working set, click the arrow on the toolbar of the Application Development view, then click **Edit Active Working Set**. When you delete the active working set, all resources are shown automatically. You cannot edit the working sets that are created automatically for an application or library, as indicated by brackets; for example, [Application1].
- To delete the selected working set, click the arrow on the toolbar of the Application Development view, click **Select Working Set**, then click **Remove**.
- To show all resources, click the arrow on the toolbar of the Application Development view, then click **Deselect Working Set**. Alternatively, click the **Remove all filters** icon  on the toolbar of the Application Development view.

Integrating the Rational Team Concert client with the IBM App Connect Enterprise Toolkit

You can integrate the Rational Team Concert client with the IBM App Connect Enterprise Toolkit.

About this task

To integrate the Rational Team Concert client with the IBM App Connect Enterprise Toolkit, complete the following steps.

The supported versions of Rational Team Concert are V4.0.3 or later, and V5.0 or later, with Eclipse 4.2.2 support.

Procedure

1. Download the Rational Team Concert client.
   a) Navigate to the Rational Team Concert download site: https://jazz.net/downloads/rational-team-concert/.
   b) Select the client that you want and click **More download options**.
      For example, select **Rational Team Concert 5.0.2** if you want the Rational Team Concert V5.0.2 client.
   c) Select **p2 Install Repository** in the **Plain .zip Files** section to select the p2 install repository.
   d) Sign in to Jazz® and locate the p2 install directory.
      For example, the p2 install repository for the Rational Team Concert V5.0.2 client is `RTC-Client-p2Repo-5.0.2.zip`.
   e) Unzip the p2 install repository into a local directory.
2. Start the IBM App Connect Enterprise Toolkit and select **Help** > **Install New Software**.
3. In the **Available Software** window, click **Add**.
4. In the **Add Repository** window, click **Archive**, navigate to the local directory in which you unzipped the p2 install repository in step "1.e" on page 109, and click **OK**.
5. Ensure that **Group items by category** is selected, and then select **Rational Team Concert Client (extend an Eclipse installation)**.
6. Click **Next**, then **Next** again in the **Install Details** window.
7. Accept the terms of the license agreement and click **Finish**.
   A security warning window opens during the installation because the Rational Team Concert client bundles and features are not signed.
8. Click **OK** to complete the installation.
9. Click **Restart Now** when prompted to restart the IBM App Connect Enterprise Toolkit.

What to do next

To work with Rational Team Concert source control:

1. Start the IBM App Connect Enterprise Toolkit.
2. Open the Work Items perspective and click **Window** > **Open Perspective** > **Other** > **Work Items**.
3. Click the **Create Repository Connection** link in the Team Artifacts view.
4. Follow the dialog and enter the information given to you by your Jazz administrator.
5. To add your project to the repository, right-click the project and select **Team** > **Share Project**.
6. In the Share Project window, select Jazz Source Control as the repository type.

See Getting started in your Rational Team Concert® source control workspace for more details about the Rational Team Concert source control operations.

You can also use the Jazz Team Build as your build engine for the IBM App Connect Enterprise Toolkit; see Building with Jazz Team Build for more information.

Configuring App Connect Enterprise to run on premises (on a physical or virtual machine)

You can create one or more integration nodes on one or more computers, and configure them on your test and production systems to process messages that contain your business data.

About this task

On Linux and Windows systems, you can install IBM App Connect Enterprise and the IBM App Connect Enterprise Toolkit, and start developing your applications. You can also configure an environment for more advanced application development and unit test, as described in “Configuring the IBM App Connect Enterprise Toolkit” on page 103.

Use the information in this section to plan and configure the resources that you need to run on premises. Alternatively, you can configure App Connect Enterprise to run in a container or in Docker.

Procedure

1. Plan your system; see “Planning an IBM App Connect Enterprise environment” on page 8.
2. Ensure that you have the correct authorization and permissions to create and access components.

   For more information, see “Authorization for configuration tasks” on page 2492 and “Security for runtime components” on page 2632.
3. Create the components, as described in “Configuring integration nodes” on page 113 and “Configuring integration servers” on page 167.
4. If you want to ensure that you are using the correct operation mode for your license, see “Checking the operation mode” on page 87 and “Changing the operation mode” on page 86.
5. Create and configure the databases.
6. If you want to ensure the data integrity of transactions, Configure global coordination of transactions.
7. If you want to connect to external resources such as Enterprise Information Systems, IMS, or JMS, Configure properties to connect to external resources.
8. If you want to configure the storage of events for aggregation, Collector, resequence, or timeout nodes, or configure monitoring event sources, Configure internal resources that are required by message flows.
9. If you want to view objects in a different language or code page, Change the locale.
10. If you want to operate your IBM App Connect Enterprise instances in a highly available configuration; see “Configuring integration nodes for high availability” on page 120.
11. If you want to operate your integration node as an IBM MQ service; see “Configuring an integration node as an IBM MQ service” on page 166.

Configuring App Connect Enterprise to run in a container

You can configure one or more integration servers and deploy them to run in a container, where they will process messages that contain your business data.

About this task

On Linux and Windows systems, you can install IBM App Connect Enterprise and the IBM App Connect Enterprise Toolkit, and start developing your applications. You can also configure an environment for more advanced application development and unit test, as described in "Configuring the IBM App Connect Enterprise Toolkit" on page 103.

Use the information in this section to plan and configure the resources that you need to run in a container. Alternatively, you can configure App Connect Enterprise to run on premises or in Docker.

If you are intending to run App Connect Enterprise in a container, you can develop and deploy your applications to an independent integration server quickly and easily, without the need to create and configure an integration node.

Procedure

Configure App Connect Enterprise to run in a container by following these steps:

1. Plan your system; see "Planning an IBM App Connect Enterprise environment" on page 8.
2. Configure security for the integration server, as described in "Configuring HTTP basic authentication for an integration node or server" on page 2548 and "Configuring SSL or TLS for an integration node or server" on page 2548.
3. Create and configure your integration servers, as described in "Configuring integration servers" on page 167.
4. If you want to ensure that you are using the correct operation mode for your license, see "Checking the operation mode" on page 87 and "Changing the operation mode" on page 86.
5. Create and configure the databases.
6. If you want to connect to external resources such as Enterprise Information Systems, IMS, or JMS, Configure properties to connect to external resources.
7. If you want to configure the storage of events for aggregation, Collector, resequence, or timeout nodes, or configure monitoring event sources, Configure internal resources that are required by message flows.
8. If you want to view objects in a different language or code page, Change the locale.

Configuring App Connect Enterprise to run in Docker

You can configure IBM App Connect Enterprise to run in a Docker container on AIX, Linux, and Windows systems, and in IBM z/OS Container Extensions (zCX).

About this task

You can use Docker to package an integration server into a standardized unit for software development. Changes to your IBM App Connect Enterprise environment can then be deployed to test and staging systems quickly and easily, which can be a major benefit to continuous delivery in your enterprise. You can choose to build specific configurations into your IBM App Connect Enterprise Docker images, such as integration servers where particular applications are deployed. Or you can choose to start a Docker container from the image, then deploy changes to the running integration server.

You can create your own Docker images by using the supplied files on ot4i (at https://github.com/ot4i/ace-docker). Alternatively, you can download a pre-built image of the IBM App Connect Enterprise for Developers edition (on selected platforms) to run in a Docker container. For more information, see Obtaining the IBM App Connect Enterprise server image from the IBM Cloud Container Registry. A pre-built image is not currently available for download for IBM z/OS Container Extensions (zCX).

Procedure

Use the information in the following topics to help you plan, install, and configure IBM App Connect Enterprise to run in Docker:

- "Docker support on Linux systems" on page 112
- "Stopping an IBM App Connect Enterprise Docker container" on page 112
- "Building a sample IBM App Connect Enterprise image using Docker" on page 113

What to do next

For more information about using Docker with IBM App Connect Enterprise, see Running IBM App Connect Enterprise in a Docker container.

Docker support on Linux systems

IBM App Connect Enterprise supports the use of Docker for hosting production servers, and for development and test environments. You can configure IBM App Connect Enterprise to run in a Docker container on AIX, Linux, and Windows systems, and in IBM z/OS Container Extensions (zCX).

About this task

For the latest information about version numbers and supported environments, see the system requirements information on the IBM Support web pages.

When you are running IBM App Connect Enterprise in a Docker container, you can build your own Docker images or you can use the samples that are provided. Note the following requirements:

- You must use Docker v1.7 or higher.
- The Docker guest must be an IBM App Connect Enterprise supported operating system.
- The host on which the Docker container is running must use a Linux kernel at level 3.10 or later.
- IBM App Connect Enterprise has no specific requirements for kernel configuration parameters, but other supporting products such as IBM MQ might have additional requirements.
- As advised in the Docker documentation, if you want data that is related to IBM App Connect Enterprise to be persisted independently of the container lifecycle, you must ensure that it is stored on a Docker volume. For more information, see https://docs.docker.com/engine/tutorials/dockervolumes/.
- To receive IBM support, you must have the following:
  - A means of capturing syslog messages.
  - A means of running commands against the running node; for example, by using `docker exec bash -c mqsilist` with the mqsiprofile set on login, or through `ssh`.
  - Access to the workpath directories, for diagnostic purposes.

For more information about Linux support, see IBM App Connect Enterprise system requirements for Linux systems:

Stopping an IBM App Connect Enterprise Docker container

You can stop your IBM App Connect Enterprise Docker container by running the `docker stop` command or by pressing `Ctrl-C`. You can configure IBM App Connect Enterprise to run in a Docker container on AIX, Linux, and Windows systems, and in IBM z/OS Container Extensions (zCX).

About this task

You can stop a Docker container by using one of the following methods:

- If the container is running with a pseudo-TTY allocated (by specifying the **-t** flag on the `docker run` command), you can stop it by pressing `Ctrl-C`. The signal is caught by the integration server, which shuts down gracefully.
- Run the `docker stop` command against the running container, as shown in the following example:

```
docker stop myAce
```

Building a sample IBM App Connect Enterprise image using Docker

You can build a sample IBM App Connect Enterprise Docker image. You can configure IBM App Connect Enterprise to run in a Docker container on AIX, Linux, and Windows systems, and in IBM z/OS Container Extensions (zCX).

About this task

To run IBM App Connect Enterprise in a Docker container, you must first build a base image containing an installation of IBM App Connect Enterprise. Instructions are available in the IBM App Connect Enterprise GitHub repository, at https://github.com/ot4i/ace-docker. Alternatively, you can download an image of the IBM App Connect Enterprise for Developers edition (for selected platforms) from https://hub.docker.com/r/ibmcom/ace/ and use this to run a Docker container.

If you plan to use IBM z/OS Container Extensions (zCX), ensure that you have installed the Linux on zSeries package. Until APAR OA59111 is available, you can build an image on IBM z/OS Container Extensions (zCX) only by using the experimental Ubuntu Dockerfiles. A pre-built image is not currently available for download for IBM z/OS Container Extensions (zCX).

A Dockerfile is a set of instructions for building a Docker image. Images can be stored in local or remote registries, and are used to create a running Docker container. To build the image, Docker executes the instructions in the Dockerfile. Each instruction causes a new image layer to be created. Docker best practice guidelines recommend that you keep the number of Dockerfile instructions to a minimum, because the number of layers in an image might be limited. The guidelines, which are available at https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices, suggest that you “find the balance between readability (and thus long-term maintainability) of the Dockerfile and minimizing the number of layers it uses. Be strategic and cautious about the number of layers you use.”

The following GitHub repository contains a Dockerfile and some additional files that show how you can build an IBM App Connect Enterprise Docker image: https://github.com/ot4i/ace-docker.

If you do not have access to the GitHub CLI (which is not currently available on IBM z/OS Container Extensions (zCX)), you can obtain the contents of the repository by running the following curl command:

```
curl -Lk https://github.com/ot4i/ace-docker/tarball/master | tar xz
```

Configuring integration nodes

Create and configure the integration node that you want on the operating system of your choice.

Before you begin

Ensure that the following requirements are met:

- Your user ID has the correct authorizations to perform the task. The authorizations are defined in Security requirements for administrative tasks.
- Windows On Windows: you have created a user ID to be used as the service user ID. You specify this ID when you create the integration node; it is used to run the integration node.

  For more information about user ID authorization and creation, refer to “Planning for security” on page 17.

- You have initialized the command environment on distributed systems; see “Setting up a command environment” on page 64.

About this task

When you have created your physical components, you can administer your test and production environments programmatically by using the IBM Integration API.

This collection of tasks uses specific resource names and user IDs. These names are examples only; you can use your own names. Follow existing naming conventions for IBM MQ and other resources.

Procedure

- Create an integration node by following in the instructions in “Creating an integration node” on page 114.
- Verify that the integration node is created by following the instructions in “Verifying an integration node” on page 120.
- Modify the integration node by following the instructions in “Configuring an integration node by modifying the node.conf.yaml file” on page 118.
- (Optional) If you are developing message flows that include resources such as the WebSphere Adapters nodes, IMS nodes, or the CICSRequest node, you must set up your environment to support these applications. Details of the setup that you require is in “Configuring the environment to access external applications and resources” on page 182.
- (Optional) If you are working with web services message flows and want to ensure reliable messaging, you can configure WS-RM using policy sets. See “Web Services Reliable Messaging” on page 818.
- (Optional) You can use WS-Security with your web services message flows to provide quality of protection through message integrity, message confidentiality, and single message authentication. See “WS-Security” on page 2650.

Creating an integration node

You can create integration nodes on every software platform that is supported by IBM App Connect Enterprise.

Before you begin

Complete the following tasks:

- Ensure that your user ID has the correct authorizations to perform the task. Refer to Security requirements for administrative tasks.
- On distributed systems, you must set up your command line environment before you create an integration node by running the product profile or console; refer to “Setting up a command environment” on page 64.

About this task

Create an integration node by using the command line on the computer on which you installed IBM App Connect Enterprise.

You must give the integration node a name that is unique on the local computer. Integration node names are case-sensitive on all supported platforms except Windows.

Optionally, on all distributed systems you can specify a queue manager to be associated with the integration node. This queue manager is used by default for IBM MQ processing in the message flow if no queue manager has been specified explicitly on the MQ node. Some message flow nodes such as CD and FTE nodes, and event-driven processing nodes used for aggregation, timeout, message collections, and message sequences, require a queue manager to be specified for the integration node. These nodes use system queues, which are owned by the queue manager, to store information about in-flight messages.

For information about creating the system queues, see “Creating the default system queues on an IBM MQ queue manager” on page 99.

Queues that are specified on event-driven processing nodes are created automatically when the flow is deployed, provided that the integration node has the required permissions to create the queues. If the queues are not created automatically, you can create them manually; see “Creating the default system queues on an IBM MQ queue manager” on page 99. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96 and “Enhanced flexibility in interactions with IBM MQ ” on page 26.

Multiple integration nodes can specify the same queue manager. You can also specify a queue manager to be used by an existing integration node, by modifying the `node.conf.yaml` configuration file for the integration node. You must then restart the integration node for the changes to take effect.

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers and message flows that you can deploy. For more information, see “Restrictions that apply in each operation mode” on page 5.

To create an integration node, complete the steps in “Creating an integration node by using the command line” on page 115.

If you want to configure the integration node as an IBM MQ trusted application, see “Using IBM MQ trusted applications” on page 117.

Creating an integration node by using the command line

On Linux, UNIX and Windows, you can create integration nodes on the command line.

Before you begin

- If you want to configure the integration node as an IBM MQ trusted application, see “Using IBM MQ trusted applications” on page 117.
- Read “Controlling access to IBM App Connect Enterprise” on page 2633.
- Check which operation mode you are licensed to use. If you do not set a mode, the automatic default is `advanced` mode; see “Operation modes” on page 3.

About this task

When you create an integration node, you can optionally specify a queue manager for the integration node. If you do not specify a queue manager, features that require access to IBM MQ will be unavailable. For more information about using IBM MQ with IBM App Connect Enterprise, see “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “Installing IBM MQ” on page 96.

When you create an integration node, a subdirectory is created to store files for the integration node (and its integration servers and the resources deployed to those servers). These files include the integration node's configuration file, `node.conf.yaml`. By default, the node's subdirectory is created under the IBM App Connect Enterprise working directory, which was set when the product was installed. On the **`mqsicreatebroker`** command you can optionally use the **`-w`** *workPath* parameter to specify your own choice of working directory.

For example, for the integration node *node_name*, the default working directory on Windows is `C:\ProgramData\IBM\MQSI\components\`*node_name*, which has the following structure:

```
C:\ProgramData\IBM\MQSI\components\node_name
C:\ProgramData\IBM\MQSI\components\node_name\node.conf.yaml
...
C:\ProgramData\IBM\MQSI\components\node_name\servers
```

The *node_name*`\servers` directory is created when you later create an integration server for the integration node, and a subdirectory is created under *node_name*`\servers` to store files for each integration server. These files include the server's configuration file, `server.conf.yaml`. For example, two integration servers created under the integration node `NODEIPL102`:

```
C:\ProgramData\IBM\MQSI\components\NODEIPL102\servers\N101ISAA
```

```
C:\ProgramData\IBM\MQSI\components\NODEIPL102\servers\N102ISAB
```

When you create an integration node by running the **mqsicreatebroker** command, an overrides subdirectory is also created under the working directory for the integration node. This overrides directory contains an additional `node.conf.yaml` configuration file, which contains property values that are set by IBM App Connect Enterprise commands, including the **mqsicreatebroker** command. These values override any values that are set for the same properties in the integration node's base `node.conf.yaml` file.

When you have created the integration node, you can configure it by modifying properties in its `node.conf.yaml` configuration file (for example, `C:\ProgramData\IBM\MQSI\components\acev11node\node.conf.yaml`). If any commands are run subsequently, which modify settings for the integration node, those modified settings are saved in a `node.conf.yaml` file in the overrides directory (for example, `C:\ProgramData\IBM\MQSI\components\acev11node\overrides\node.conf.yaml`).

If a property has been set in the integration node's base `node.conf.yaml` file, and also in the overrides directory (`\overrides\node.conf.yaml`), the property value that has been set in the overrides directory is used. Therefore, if an integration node does not appear to be using the settings that you would expect, check the `node.conf.yaml` file in the overrides directory to see if your expected property value has been overridden by a command. If you want to manually override the settings that have resulted from a command, you can either edit the property in the `node.conf.yaml` file in the overrides directory, or you can remove the entry from the overrides directory and modify the base `node.conf.yaml` file instead.

To create an integration node by using the command line, complete the following steps:

Procedure

1. Ensure that you have the authority to create an integration node by following the steps for your operating system:
   - Linux UNIX On Linux and UNIX, ensure that you are logged in using a user ID that has authority to run the **mqsicreatebroker** command.

     Run the **mqsiprofile** script to set up the command environment for the integration node:

     ```
     install_dir/server/bin/mqsiprofile
     ```

     You must run this script before you can run the IBM App Connect Enterprise commands.

     For more information, see “Setting up a command environment” on page 64.
   - Windows On Windows, open an IBM App Connect Enterprise command prompt for the runtime installation in which you want to create the integration node. For more information about initializing the runtime environment, see “Setting up a command environment” on page 64.

     On Windows systems, you must open a command console with elevated privileges. To open a command console with elevated privileges, use the **mqsicommandconsole** command. For more information, see command.
2. Use the **mqsicreatebroker** command to create the integration node.

   For example, if you want to create an integration node that is called INODE with a queue manager that is called ACEQMGR, enter the following command:

   ```
   mqsicreatebroker INODE -i wbrkuid -a wbrkpw -q ACEQMGR
   ```

   Where wbrkuid and wbrkpw are the user name and password under which the integration node runs.

   Note: You can specify a queue manager name that does not exist, but the specified queue manager must be running before you start your integration node.

   When you create your first integration node, the web user interface uses the default port 4414, and for each integration node created after the port number increments automatically by one. You can

change the port number to be used by editing the integration node's `node.conf.yaml` file. You can also change the port number or disable the web user interface by using the **mqsichangeproperties** command. If administration security is not enabled, web users can access the web user interface as a default user with unrestricted access to data and integration node resources.

For more information about the command parameters, see command.

Results

You have created an integration node.

What to do next

(Optional) If you want to configure properties of the integration node, edit its `node.conf.yaml` file. For example, you can set a REST administration port and an HTTPS port, you can enable administration security, and you can configure the trace level, activity logging, JVM, and the reporting of statistics and accounting data. For more information, see "Configuring an integration node by modifying the node.conf.yaml file" on page 118.

Start the integration node by using the **mqsistart** *node_name* command. See "Starting and stopping an integration node on Windows, Linux, and UNIX systems" on page 247.

1. If you want to work with the integration node in the IBM App Connect Enterprise Toolkit (or a custom integration application), connect the toolkit to the integration node. For example, connect to the integration node before you can use the Enterprise Toolkit to start the node's web user interface or create integration servers associated with the integration node. See "Connecting to an integration node by using the Toolkit" on page 245.
2. If you want to use the web user interface to manage the integration node's integration servers and their resources, start the web user interface using the administration URI (hostname and port) configured for the integration node. The hostname and port are reported by the **mqsilist** command; for example:

   ```
   mqsilist
   ...
   BIP1325I: Integration node 'NODEIPL101' with administration URI 'http://<host>:4417' is
    running.
   ```

   For more information about using the web user interface to manage integration servers and their resources, see "Managing integration servers" on page 249 and "Managing deployed resources" on page 316.
3. Create integration servers under the integration node; for example, by using the Enterprise Toolkit or the web user interface. For more information, see "Creating an integration server" on page 168.

When you have completed these tasks, you can create the resources that you want to associate with the integration node; for example message flows. You can create and work with resources by using either the IBM App Connect Enterprise Toolkit or the IBM Integration API.

Using IBM MQ trusted applications

Configure an integration node to run as an IBM MQ trusted application.

Before you begin

You must complete the following tasks:

- Ensure that your user ID is a member of the mqm group. On UNIX and Linux, specify the user ID mqm as the service user ID when you create the integration node. On Windows, use any service user ID that is a member of mqm. Refer to Security requirements for administrative tasks.
- Review the restrictions that IBM MQ places on trusted applications that apply to your environment. See Connecting to a queue manager using the MQCONNX call in the IBM MQ product documentation online.

About this task

You can configure an integration node to run as a trusted (fastpath) application on all supported platforms . If the integration node is configured as a trusted application, it runs in the same process as the IBM MQ queue manager agent, and all integration node processes benefit from an improvement in the overall system performance.

An integration node does not run as a trusted application by default; therefore, you must create a trusted application by using the command.

Configuring an integration node as a trusted application does not affect the operation of IBM MQ channel agents or listeners. For more information about running these as trusted applications, see Running channels and listeners as trusted applications in theIBM MQ product documentation online.

Take care when deploying user-defined message flow nodes or parsers. Because a trusted application (the integration node) runs in the same operating system process as the queue manager, a user-defined message flow node or parser might compromise the integrity of the queue manager. Consider fully the restrictions that apply to your environment and test user-defined nodes and parsers in a non-trusted environment before deploying them in a trusted integration node.

You can configure an integration node to run as a trusted application when you create it.

Procedure

- To create an integration node on a command line, run the **mqsicreatebroker** command with the `-t` flag, which specifies that the integration node is created as a trusted application.

  For example, enter the following command to create an integration node called INODE as a trusted application:

  ```
  mqsicreatebroker INODE -q ACEQMGR -i mqm -t
  ```

  See “Creating an integration node” on page 114 for more detailed information about how to create an integration node for your platform.

Configuring an integration node by modifying the node.conf.yaml file

You can configure an integration node by setting properties in the `node.conf.yaml` configuration file for the integration node.

Before you begin

You must have completed the following tasks:

- Ensure that your user ID has the correct authorizations to perform the task; see Security requirements for administrative tasks.
- Create an integration node.

About this task

When you create an integration node, a default `node.conf.yaml` configuration file is created automatically for the integration node and stored in the working directory for the integration node; for example: `C:\ProgramData\IBM\MQSI\components\`*`acev11node`*`\node.conf.yaml`. You can then configure the operation of the integration node and associated resources, by modifying properties in the `node.conf.yaml` file. For example, you can set a REST administration port and an HTTPS port, you can enable administration security, and you can configure the trace level, activity logging, JVM, and the reporting of statistics and accounting data.

When you create an integration node, an overrides subdirectory is also created under the integration node's working directory. This overrides directory contains an additional `node.conf.yaml` configuration file, which contains property values that are set by IBM App Connect Enterprise commands, including the

mqsicreatebroker command. These values override any values that are set for the same properties in the integration node's base `node.conf.yaml` file.

When commands are run that modify settings for the integration node, those modified settings are saved in a `node.conf.yaml` file in the overrides directory (for example, `C:\ProgramData\IBM\MQSI\components\`*`ace11node`*`\overrides\node.conf.yaml`).

If a property has been set in the integration node's base `node.conf.yaml` file, and also in the overrides directory (`\overrides\node.conf.yaml`), the property value that has been set in the overrides directory is used. Therefore, if an integration node does not appear to be using the settings that you would expect, check the `node.conf.yaml` file in the overrides directory to see if your expected property value has been overridden by a command. If you want to manually override the settings that have resulted from a command, you can either edit the property in the `node.conf.yaml` file in the overrides directory, or you can remove the entry from the overrides directory and modify the base `node.conf.yaml` file instead.

When you create integration servers that are owned by the integration node, a default `server.conf.yaml` configuration file is created for each of the integration servers, and they are stored in the file system in subdirectories below the integration node directory. Any properties that you set for the integration node, in the `node.conf.yaml` file, are inherited by the integration servers that it owns. However, you can change any of the integration server properties by modifying them in the appropriate `server.conf.yaml` file. For more information about configuring integration servers that are managed by an integration node, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172.

Procedure

Change the properties of an integration node, by following these steps:

1. Use a YAML editor to open the `node.conf.yaml` file for the integration node that you want to modify.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not accepted in YAML and will cause your integration node configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Modify the properties that you want to change:
   a) Set the administration REST API port for the App Connect Enterprise web user interface and the App Connect Enterprise Toolkit, by setting a value for the **port** property. You can leave this property set to the default value of 4414.
   b) Specify a value for the **host** property.
   c) Enable authentication for the integration node, by setting the **basicAuth** property to `true`:

   ```
   basicAuth: true
   ```

   d) Enable administration security and specify an authorization mode, by setting the **authorizationEnabled** and **authorizationMode** properties.
   For example:

   ```
   authorizationEnabled: true
   authorizationMode: 'file'
   ```

   e) You can also modify properties to enable the integration node listener, to configure the MQTT server, and to specify a default queue manager.
3. Specify the security credentials to be used by the integration node when connecting to a secured resource (such as a database) by using the **mqsisetdbparms** command. When you run this command, the user ID and password are stored securely in the IBM App Connect Enterprise credentials store. For more information about using this command, see command.

4. Restart the integration node.

   The properties that you set in the `node.conf.yaml` file take effect when the integration node is started. If you modify these properties again, you must start the integration node again for the latest changes to take effect. For more information, see "Starting and stopping an integration node" on page 247.

Verifying an integration node

Use the **mqsilist** command to display the status of integration nodes.

About this task

If you run the **mqsilist** command with no parameters, that command displays a one line summary for every integration node in the current version of IBM App Connect Enterprise. For example:

```
mqsilist
...
BIP1325I: Integration node 'ACE01NODE' with administration URI 'http://localhost:4414' is
 running.
BIP1326I: Integration node 'ACE02NODE' is stopped.
BIP8071I: Successful command completion.
```

You can request further details about all resources by specifying the parameter **-d 2** on the command. You can also specify the **-r** parameter so that the command recursively returns information about integration servers, and the message flows and files that you have deployed to those integration servers.

You can also use this command to display a list of integration nodes that you have created for all concurrent installations on this computer. For example, you might have installed both Version 10.0 and Version 12.0 for assessment and migration. For example:

```
mqsilist -a
...
BIP1325I: Integration node 'ACE01NODE' with administration URI 'http://localhost:4414' is
 running.
BIP1326I: Integration node 'ACE02NODE' is stopped.
BIP8221I: Broker: IB01NODE (Version 10.0)  -
BIP8071I: Successful command completion.
```

Configuring integration nodes for high availability

If you want to operate your IBM App Connect Enterprise instances in a highly available configuration, you can set up your integration nodes to work with IBM MQ multi-instance queue managers, a replicated data queue manager (RDQM), the HTTP proxy servlet, an existing Windows Cluster (Windows Server), or a high availability manager such as HACMP, HA/XD, or Veritas Cluster Server (VCS).

About this task

The following topics describe how to configure your integration node for high availability:

- "Configuring multi-instance integration nodes" on page 121
- "Configuring the HTTP proxy servlet" on page 149
- "Using an integration node with an RDQM configuration" on page 139
- "Using an integration node with an existing high availability manager" on page 130
- "Using an integration node with an existing Windows Cluster (Windows Server)" on page 138

High availability overview

An overview of the high availability configuration and topology options available in IBM App Connect Enterprise.

In IBM App Connect Enterprise, you can create a high availability solution by using any of the following methods:

Multi-instance integration nodes with IBM MQ
Use a multi-instance integration node so that the integration node is available in the same location as the IBM MQ multi-instance queue manager in a high availability IBM MQ configuration. The multi-instance integration node can either dynamically start in all locations where the multi-instance queue manager runs, or can be set as an MQ Service dependency.

For more information, see "Configuring multi-instance integration nodes" on page 121.

An existing high availability manager
Use an external high availability manager by mounting the external resources to the shared file system. Script files are provided for the common tasks that are required to create a multi-instance integration node.

For more information, see "Using an integration node with an existing high availability manager" on page 130.

An RDQM configuration
Configure an integration node to work with an IBM MQ replicated data queue manager (RDQM).

For more information, see "Using an integration node with an RDQM configuration" on page 139.

An existing Windows Cluster
Add an integration node to the nodes in a Windows Cluster.

For more information, see "Using an integration node with an existing Windows Cluster (Windows Server)" on page 138.

The HTTP proxy servlet
Use the HTTP proxy servlet in a servlet container so that you can support high availability, load distribution, access to the integration node from multiple IP addresses and ports, and a larger number of concurrent HTTP sessions.

For more information, see HTTP proxy servlet overview.

Configuring multi-instance integration nodes

Use IBM MQ to configure an integration node to run in multi-instance mode for high availability.

Before you begin

- Review the topic Multi-instance queue managers in the IBM MQ product documentation to understand the use of multi-instance queue managers.

About this task

The set of tasks that describe how to configure a multi-instance integration node assumes the following criteria:

- A file server is configured to host both the shared work path directory for the multi-instance integration node and the shared directories for the multi-instance queue manager.
- The file server is shared between two computers, each of which has a licensed copy of the IBM App Connect Enterprise and IBM MQ products installed.

The steps apply to all operating systems that are supported by IBM App Connect Enterprise and IBM MQ.

To configure an integration node to run in multi-instance mode, complete the following steps.

Procedure

1. Create a shared path directory on an NFS or NAS server.

   On Windows, you can use a shared UNC path. Follow the steps as described in the IBM MQ product documentation; see Creating a shared file system.

2. Create or configure the multi-instance queue manager. Follow the appropriate steps for your operating system as described in the IBM MQ product documentation; see Create a multi-instance queue manager.
3. Follow the steps to create the multi-instance integration node as described in "Creating a multi-instance integration node" on page 122.

What to do next

After you have created a multi-instance integration node, you can also optionally complete the following actions.

- Use the **mqsilist** command to view integration nodes, and which integration nodes are multi-instance.
- Delete a multi-instance integration node, as described in "Deleting a multi-instance integration node" on page 125.
- Delete a multi-instance queue manager, as described in the IBM MQ product documentation, Deleting a multi-instance queue manager.
- Verify shared file system locks, as described in the IBM MQ product documentation, Verifying shared file system behavior.
- Back up, and later restore a multi-instance integration node, as described in "Backing up and restoring a multi-instance integration node" on page 128
- Learn how to fail over multi-instance integration nodes, as described in "Failing over a multi-instance integration node and queue manager" on page 129.

Creating a multi-instance integration node

If you specify a multi-instance queue manager for an integration node, you must configure that integration node as multi-instance.

Before you begin

- Create the shared directories that you require for the multi-instance integration node, as described in the topic Create a multi-instance queue manager in the IBM MQ product documentation. The examples shown in this topic assume that you have created a shared directory called `MQHA`.
- Create the IBM MQ multi-instance queue manager. On Windows, the queue manager must be created with the `"-a"` or `"-ar"` flag on **crtmqm**, specifying a domain group that IBM MQ can use for securing shared files. If you have the choice, use the `"-ar"` flag. For more information, see Create a multi-instance queue manager in the IBM MQ product documentation.

About this task

You can use two configurations for a multi-instance integration node:

- Configure the multi-instance integration node with explicit instances where the queue manager can run. The multi-instance integration node runs in all the defined locations where the multi-instance queue manager is available, and is inactive in locations where the queue manager is not running.
- Configure the multi-instance integration node as an MQ Service dependency. When a multi-instance integration node depends on an MQ Service, whenever the multi-instance queue manager becomes unavailable, the integration node stops. When the queue manager starts, the integration node is also started on the same computer that the queue manager is running on.

To create a multi-instance integration node of either configuration, complete the following steps:

Procedure

1. On the computers that will run the instances of the integration node, configure the required users and groups so that they will have access to the directory for the shared file system:
   - Linux UNIX On Linux and UNIX, the uid and gid for the mqbrkrs group in `/etc/password` must be the same on each server. For more information, see Create a multi-instance queue manager in the IBM MQ product documentation.
   - Windows On Windows, create the following users and groups:
     a. A domain group that is a member of the local mqbrkrs group on both systems. For example, `ACE\Domain mqbrkrs`.
     b. A domain user that is a member of the Domain mqbrkrs and mqm groups. This ID is used for running the integration node.
     c. A domain user that is a member of the Domain mqbrkrs group and a member of the local Administrators group on both machines. This ID is used for creating the integration node. You can use the same ID for both creating and running the integration node, but you do not have to be an Administrator to run the integration node. For example, `WMB\mqsiuser-admin`. The listed user and groups are using the example domain name ACE.
2. Create a directory for the integration node shared files on the file server:
   - Linux On Linux and UNIX systems, create a subdirectory (such as `mqsi`) on the shared drive (`MQHA`); for example: `/MQHA/mqsi`. Ensure that this directory is owned by the mqbrkrs user and group, and has the access permissions `rwx`. The UID of the integration node user ID in `/etc/passwd` and the GID for mqbrkrs in `/etc/group` must be the same on each server.
   - Windows On Windows, update the security permissions of the folder:
     a. In Windows Explorer, right-click the shared directory that you created, and select **Properties.**
     b. Click the **Security** tab, then click **Advanced** > **Change Permissions...**
     c. Clear **include inheritable permissions from this objects parent**.
     d. In the **Permission entries** window, select the entries for individual users and then click **Remove**. Leave the entries for SYSTEM, Administrators, and CREATOR OWNER.
     e. Add `mqbrkrs` with **Full Control**. If this folder is also being used for multi-instance queue manager, then the domain group that is used to secure the queue manager must also be added with **Full Control** set.
     f. Add the global group domain `mqm`. Click **Check Names**, and then click **OK**. If this folder is also being used for multi-instance queue manager, then the domain group that is used to secure the queue manager must also be added with **Full Control** set.
     g. Remove the default **Everyone** user from the list.
3. Open the command console or command prompt by running one of the following commands:
   - Linux UNIX On Linux and UNIX systems, as a non-root user that is in the mqm and mqbrkrs groups, open a command prompt, navigate to the `bin` directory of the installation, and run the **`mqsiprofile`** command. For example:

     ```
     . /opt/ibm/ace-11.0.0.n/server/bin/mqsiprofile
     ```

   - Windows On Windows, as the user `mqsiuser-admin`, open a command prompt with elevated privileges by using the **`mqsicommandconsole`** command; see command

4. Create a multi-instance integration node on computer A, by running one of the following commands:

   - Linux On Linux and UNIX systems:

     ```
     mqsicreatebroker INODE -q QM1 -e /MQHA/mqsi
     ```

     where INODE is the name of the integration node and the **-e** parameter specifies the name of the shared directory created in step 2.

   - Windows On Windows:

     ```
     mqsicreatebroker INODE -i "WMB\mqsiuser" -a password -q QM1 -e \\MQHA\\mqsi -B "WMB\Domain
      mqbrkrs"
     ```

     where INODE is the name of the integration node, and *password* is the `mqsiuser-admin` password. The **-e** parameter specifies the name of the shared directory that was created in step 2, and `QM1` is the name of the multi-instance queue manager that was created previously.

   If you want to start the multi-instance integration node as an MQ Service dependency, specify **-d** as `defined` on the **mqsicreatebroker** command. For more information, see command.

   Note: You must be a member of the mqm group to run the **mqsicreatebroker** command with the **-d** parameter.

   You must ensure that the shared location exists, and that your user ID has access to the shared location before you run this command.

5. Add the details of integration node INODE onto computer B as an instance of that integration node.

   Use the **mqsiaddbrokerinstance** command, in the appropriate format for your operating system.

   - Linux On Linux and UNIX:

     ```
     mqsiaddbrokerinstance INODE -e /MQHA/mqsi
     ```

     where the **-e** parameter specifies the name of the shared directory created in step 2.

   - Windows On Windows:

     ```
     mqsiaddbrokerinstance INODE -i "WMB\mqsiuser" -a password -e \\MQHA\\mqsi
     ```

     where the **-e** parameter specifies the name of the shared directory created in step 2.

   For more information, see command.

   Repeat this step for every computer that the multi-instance queue manager runs on.

6. Start queue manager `QM1` so that it is active on computer A.

   See Starting and stopping a multi-instance queue manager in the IBM MQ product documentation.

7. Start integration node `INODE` on computer A.

   Use the **mqsistart** command:

   ```
   mqsistart INODE
   ```

8. Start integration node INODE on computer B.

   You can observe that integration node INODE is running in standby mode against the standby queue manager `QM1` by running the command **mqsilist**.

9. Optional: Optional: test that the integration node works as follows:
   a) Stop integration node INODE and queue manager QM1 on computer A.

      Observe on computer B that integration node INODE and queue manager QM1 change from standby to active mode.
   b) Restart queue manager QM1 and integration node INODE on computer A.

      Observe on computer A that queue manager QM1 and integration node INODE are in standby mode and, on computer B, queue manager QM1 and integration node INODE remain in active mode.

Results

You have configured a multi-instance integration node, and created an instance of that integration node. When integration node INODE and queue manager QM1 stop on computer A, the same integration node and queue manager on computer B become active, and return to standby when computer A becomes active again.

If you chose to define the multi-instance integration node as an MQ Service dependency, then the integration node stops whenever the multi-instance queue manager becomes unavailable. The integration node is started again when the queue manager starts.

Deleting a multi-instance integration node

Delete a multi-instance integration node without affecting the shared multi-instance configuration.

About this task

Important: The order of deletion of a multi-instance integration node and its associated instances affects the shared configuration. Take care to use the correct command for each step in the process.

You must remove the integration node instances by using the **mqsiremovebrokerinstance** command before you attempt to remove the integration node itself. This command removes all local references to the integration node instance, but does not affect the shared configuration for the multi-instance integration node on the shared work path. The **mqsiremovebrokerinstance** command cannot be run against the standby integration node instance when it is running; stop the integration node instance before you run this command.

If you unintentionally remove an integration node instance, it can be re-created by using the **mqsiaddbrokerinstance** command, providing that the **mqsideletebroker** command was not already used to delete the integration node.

If the multi-instance integration node was removed by using the **mqsideletebroker** command before the associated integration node instances were removed, it will not be possible to start the integration node instances. To recover from this situation, re-create the multi-instance integration node by using the **mqsicreatebroker** command with the **-e** parameter, specifying the original shared work path location.

The following procedure gives an overview of how to delete a multi-instance integration node:

Procedure

1. Stop the multi-instance integration node on computer A.
2. Stop the integration node instance on computer B.
3. Remove the integration node instance on computer B.

   Use the following command, where INODE represents the name of the integration node:

   ```
   mqsiremovebrokerinstance INODE
   ```

   For more information, see command.

   Repeat this step for all integration node instances.

4. Remove the multi-instance integration node on computer A.

   Use the following command:

   ```
   mqsideletebroker INODE
   ```

   This process removes all references to the integration node on both the local and shared work paths. For more information, see command.

Results

You have deleted the multi-instance integration node, and removed all of its instances.

Migrating an integration node to a multi-instance integration node

Migrate an integration node to a multi-instance integration node by moving the queue manager data to a shared directory, and reconfiguring the integration node on two other servers.

Before you begin

Multi-instance integration nodes require a multi-instance queue manager. You must first check the prerequisites for running a multi-instance queue manager, and migrate your queue manager if it is single-instance. For more information, see Migrating from a single instance to a multi-instance queue manager in the IBM MQ product documentation.

For the latest information about tested environments, see Testing statement for IBM MQ multi-instance queue manager file systems. This support statement contains detailed version and prerequisite information for each environment. A test tool is provided with IBM MQ to assist you in testing unsupported environments.

You must provide three servers to run a multi-instance integration node, which provide the following functions:

- A shared file system to store the integration node data and logs.
- Active instances of the integration node.
- Standby instances of the integration node

For more information, see Create a multi-instance queue manager in the IBM MQ product documentation.

Procedure

On the servers that will run the active and standby instances of the integration node, configure the required users and groups so that they have access to a shared directory on the network file system.

1. On Windows, create the following users and groups:
   a) A domain group that is a member of the local `mqbrkrs` group on both servers. For example, `WMB\Domain mqbrkrs`
   b) A domain group that is a member of the local `mqm` group on both servers. For example, `WMB\Domain mqm`
   c) A domain user that is a member of the domain `mqbrkrs` and `mqm` groups. This ID is used for running the integration node. For example, `WMB\mqsiuser`
   d) A domain user that is a member of the domain `mqbrkrs` group and a member of the local administrators group on both machines. For example, `WMB\mqsiuser-admin`. This ID is used for creating the integration node. It can be the same as the previous ID, but you do not have to run the integration node as an administrator.
2. On UNIX and Linux, the uid and gid for `mqbrkrs` in `/etc/password` must be the same on each server. For more information, see Creating a shared file system in the IBM MQ product documentation.

Create a shared directory on the network file system with the correct access permissions. A typical configuration consists of a single shared directory that contains all data for all integration nodes that use the shared file system. Each integration node creates its own data directories under the shared directory.

3. On Windows, create a directory on the file server for the files shared by the integration node, for example, `C:\mqsishare`. Update the security permissions of the directory by completing the following steps:
   a) In Windows Explorer, right-click the shared directory that you have just created and select **Properties.**
   b) Click the **Security** tab, then click **Advanced** > **Change Permissions...**
   c) Clear the "Include inheritable permissions from this object's parent" checkbox.
   d) In the Permission entries window, select the entries for individual users and click **Remove**. Leave the entries for SYSTEM, Administrators, and CREATOR OWNER.
   e) Click **Add...**, and then enter the name of the global group domain `mqm`. Click **Check Names** and click **OK**.
   f) In the **Permission Entry** window, select the **Allow** checkbox for the **Full Control** entry.
   g) Repeat the previous two steps for the global group domain `mqbrkrs`.
   h) In the **Properties** window, click the **Sharing** tab and then click **Advanced Sharing**. Select the **Share this folder** checkbox, and leave the share name as `mqsishare`. Click **Permissions.**
   i) Click **Add**, and enter the name of the global group domain `mqbrkrs`. Click **Check Names**.
   j) In the **Group or user names** panel, select the domain `mqbrkrs`. Select the **Allow** checkbox for the **Full Control** entry, and then click **Apply**.
   k) Repeat the previous two steps for the global group `Administrators`.
   l) In the **Group or user names** panel, remove the group `Everyone`.
4. On Linux and UNIX, complete the following steps:
   a) Create `/HA/mqsi` on the shared drive.

      Ensure that `/HA/mqsi` is owned by the user and group `mqbrkrs`, and has the access permissions `rwx`.
   b) If you are using an NFS v4 file server, add the following line to the `/etc/exports` file:

      ```
      /IBHA * rw,sync,no_wdelay,fsid=0)
      ```

      Start the NFS daemon by using the following command:

      ```
      /etc/init.d/nfs start
      ```

Copying the integration node data to the share is a manual procedure. Before you continue, ensure that you have backed up your integration node. For more information, see “Backing up the integration node” on page 477.

5. Stop the integration node.

Move the IBM App Connect Enterprise configuration directories.

6. On Windows, run the following commands:
   a) **mkdir** *<server_name>*\mqsishare\mqsi\

      where *<server_name>* is the name of your server.
   b) **xcopy** /e /i /q /z C:\ProgramData\IBM\MQSI\components*<int_node_name>* \
      <server_name>\mqsishare\mqsi\components*<int_node_name>*

      where *<int_node_name>* is the name of your integration node.
   c) **xcopy** /e /i /q /z C:\ProgramData\IBM\MQSI\registry*<int_node_name>* \
      <server_name>\mqsishare\mqsi\registry*<int_node_name>*
   d) **rmdir** C:\ProgramData\IBM\MQSI\components*<int_node_name>* /q /s
   e) **rmdir** C:\ProgramData\IBM\MQSI\registry*<int_node_name>* /q /s

7. On Linux and UNIX, run the following commands:
   a) **mkdir** -p /HA/mqsi/components/
   b) **mkdir** -p /HA/mqsi/registry/
   c) **mv** /var/mqsi/components/*<int_node_name>*\ /HA/mqsi/components/
   d) **mv** /var/mqsi/registry/*<int_node_name>*\ /HA/mqsi/registry/

Create a file called HASharedWorkPath that contains the location of the new shared directory and place it with the integration nodes registry directory. The file must not contain any newline characters.

8. On Windows, run the following commands:
   a) **mkdir** C:\ProgramData\IBM\MQSI\registry*<int_node_name>*
   b) <nul set /p =*<server_name>*\mqsishare\mqsi> C:\ProgramData\IBM\MQSI \registry*<int_node_name>*\HASharedWorkPath.dat
9. On Linux and UNIX, run the following commands:
   a) **mkdir** /var/mqsi/registry/*<int_node_name>*
   b) **chmod** 770 /var/mqsi/registry/*<int_node_name>*
   c) **echo** –n /HA/mqsi> /var/mqsi/registry/*<int_node_name>*/HASharedWorkPath

On the standby server, add the standby instance using **mqsiaddbrokerinstance**. For more information, see command.

10. Configure the standby integration node:
    a) On Windows, run the following command: **mqsiaddbrokerinstance** *<int_node_name>* -i *<user_name>* -a *<password>* -e *<server_name>*\mqsishare
    b) On Linux and UNIX, run the following command: **mqsiaddbrokerinstance** *<int_node_name>* -e /HA/
11. On the active server, run the following command: **mqsistart** *<int_node_name>*
12. On the standby server, run the following command: **mqsistart** *<int_node_name>*

Results

Your single-instance integration node has been converted to a multi-instance integration node. You can verify this by using the **mqsilist** command. See command.

Backing up and restoring a multi-instance integration node

Back up the registry and configuration of the multi-instance integration node and its associated resources, and re-create that integration node from the backup.

Before you begin

To ensure that the backup is complete and correct, back up the integration node when the integration node is stopped, or when it is not processing a configuration change (such as a deployment or property change).

About this task

When you back up a multi-instance integration node, you back up the registry and configuration from the shared work path of the integration node and its associated instances. Therefore, the backup procedure that is described in this topic applies to the multi-instance integration node only. No additional steps exist for backing up the associated instances of the integration node.

When you restore a multi-instance integration node, you re-create associated instances by using the **mqsiaddbrokerinstance** command.

Procedure

1. Back up a multi-instance integration node by using the **mqsibackupbroker** command.
   For example:

   ```
   mqsibackupbroker INODE -d /BackupDirectory/ACE
   ```

   where:

   - INODE is the name of the integration node.
   - /BackupDirectory/ACE is the directory in which the backup file is created.

   The command detects a multi-instance integration node and backs up the registry and configuration from the shared work path of the integration node.
2. Restore a multi-instance integration node by using the **mqsirestorebroker** command.

   ```
   mqsirestorebroker INODE -d /BackupDirectory/ACE -a 20091009.zip
   ```

   where:

   - INODE is the name of the integration node.
   - /BackupDirectory/ACE is the directory in which the backup file is stored.
   - 20091009.zip is the name of the backup (archive) file.

   The command detects a multi-instance integration node and restores the registry and configuration to the shared work path of the integration node.
3. Re-create associated instances of the multi-instance integration node by using the **mqsiaddbrokerinstance** command.

   For more information, see the command.

Failing over a multi-instance integration node and queue manager

An active integration node instance fails over when its associated active multi-instance queue manager either terminates unexpectedly, or stops in a controlled manner. The action of stopping an active integration node instance on an active multi-instance queue manager does not by itself cause a standby integration node instance to become active.

About this task

The following examples list the 3 failover scenarios, and how to failover:

Procedure

- Controlled failover.
  - Switch over to a standby instance by stopping an active queue manager with the **endmqm** command:

    ```
    endmqm -s QueueManager
    ```

    where *QueueManager* is the name of the queue manager that you are stopping. As the active instance of the queue managers goes down, the standby instance starts.
- Immediate failover.
  - Stop an active queue manager process with the **kill** command on Linux and UNIX, or end the process with the Windows task manager on Windows:

    ```
    kill -9 amqzxma0 ProcessID
    ```

    where *ProcessID* is the process ID of the amqzxma0 process that you need to kill. The standby instance of the queue manager becomes active.

- Shutting down the server.
  - Restart the server that the active instance of the queue manager is running on. The standby instance of the queue manager becomes active.

Results

If the multi-instance integration node was configured as dependent on an MQ Service, when the multi-instance queue manager restarts, the integration node instance will also start on the computer on which the queue manager is running. Otherwise, the integration node that is in standby node becomes active.

Using an integration node with an existing high availability manager

You can use IBM App Connect Enterprise with an existing high availability manager, for example HACMP, HA/XD, Veritas Cluster Server (VCS), or HP-UX Serviceguard.

About this task

The provision of multi-instance nodes facilitates the configuration of IBM App Connect Enterprise with a high availability (HA) manager.

This topic summarizes how to complete the following tasks:

1. Create an integration node.
2. Add an integration node instance.
3. Start an integration node.
4. Stop an integration node.
5. Monitor an integration node.
6. Delete an integration node.

Procedure

1. To create an integration node, mount the shared resource onto your primary node and use the following **mqsicreatebroker** command with the **-e** parameter to specify your shared resource location.

   ```
   mqsicreatebroker INODE -e /MQHA/MyIntNode/
   ```

   where:

   - INODE is the name of the integration node.
   - /MQHA/INODE/ is the directory for your shared resource.
2. To add another integration node instance, mount the shared resource onto your secondary nodes and use the following **mqsiaddbrokerinstance** command.

   ```
   mqsiaddbrokerinstance INODE -e /MQHA/MyIntNode/
   ```

   where:

   - INODE is the name of the integration node.
   - /MQHA/INODE/ is the directory for your shared resource.
3. To start an integration node, you can use one of the following script files:

   hamqsi_start_broker

   ```
   #!/bin/ksh
   # Module:
   #   hamqsi_start_broker
   #
   # Args:
   #   BROKER = name of integration node to start
   #
   ```

```
# Description:
#   This script attempts to start the MQSI integration node
#
#   Runs as the userid which runs integration node, and must have
#   the user's environment (i.e. invoke from "su - $MQUSER ..")
#

BROKER=$1

if [ -z "$BROKER" ]
then
  echo "hamqsi_start_broker: ERROR! No integration node name supplied"
  echo "   Usage: hamqsi_start_broker <BROKER>"
  exit 1
fi

# Ensure that the integration node is not already running. In this test
# we look for any integration node-related processes, which might have
# been left around after a previous failure. Any that remain
# must now be terminated. This is a severe method of cleaning
# up integration node processes.
# We stop the processes in the following order:
#   bipservice       - first so it cannot issue restarts
#   bipbroker        - next for same reason
#   biphttplistener
#   bipMQTT
#   startDataFlowEngine
#  DataFlowEngine  - last
#
echo "hamqsi_start_broker: Ensure $BROKER not already running"
for process in bipservice bipbroker biphttplistener startDataFlowEngine DataFlowEngine
do
  # Output of kill redirected to /dev/null in case no processes
  ps -ef | grep "$process $BROKER" | grep -v grep | \
     awk '{print $2}'| xargs kill -9 > /dev/null 2>&1
done


# Start the integration node
echo "hamqsi_start_broker: Start integration node " $BROKER
mqsistart $BROKER > /dev/null 2>&1
if [ $? -ne "0" ]
then
  echo "hamqsi_start_broker: Bad result from mqsistart for $BROKER"
  exit 1
fi

# Check to see if the integration node service has started. This loop
# uses a fixed online timeout of approx. 10 seconds.
TIMED_OUT=yes
i=0
while [ $i -lt 10 ]
do
  # Check for integration node start. We look for bipservice and
  # bipbroker to be running; there might be no message flows
  # deployed.
  # Look to see whether bipservice is running
  cnt=`ps -ef | grep "bipservice $BROKER" | grep -v grep | wc -l`
  if [ $cnt -gt 0 ]
  then
    # Look to see whether bipbroker is running
    cnt=`ps -ef | grep "bipbroker $BROKER" | grep -v grep | wc -l`
    if [ $cnt -gt 0 ]
    then
      # Integration node is online
      echo "hamqsi_start_broker: ${BROKER} is running"
      TIMED_OUT=no
      break  # out of timing loop
    fi
  fi
  # Manage the loop counter
  i=`expr $i + 1`
  sleep 1
done

# Report error if integration node failed to start in time
if [ ${TIMED_OUT} = "yes" ]
then
  echo "hamqsi_start_broker: Integration node service failed to start: " $BROKER
  exit 1
fi
```

```
exit 0
```

hamqsi_start_broker_as

```
#
#!/bin/ksh
# Module:
#   hamqsi_start_broker_as
#
# Args:
#   integration node = integration node name
#   qm     = name of integration node queue manager
#   mquser = user account under which QM and Integration node are run
#
# Description:
#   Starting an MQSI integration node requires the following services:
#   (1) The MQSeries Queue Manager which supports the integration node
#   (2) The MQSI integration node service
#   This script provides a single source to initiate the required
#   services in sequence.
#
#   Queue Manager:
#   This script uses the strmqm script supplied by MQSeries V7
#
#   Integration node:
#   This script then invokes the hamqsi_start_broker script which
#   checks that the integration node is fully stopped and then starts it.
#
#   The hamqsi_start_broker_as script should be run as root.



# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi


BROKER=$1
QM=$2
MQUSER=$3

# Check all parameters exist

if [ -z "$BROKER" ]
then
  echo "hamqsi_start_broker_as: ERROR! No integration node name supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_start_broker_as: ERROR! No queue manager name supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_start_broker_as: ERROR! No Userid supplied"
  echo "  Usage: hamqsi_start_broker_as <BROKER> <QM> <MQUSER>"
  exit 1
fi

# ------------------------------------------------------------------
# Start the Queue Manager
#
echo "hamqsi_start_broker_as: Start Queue manager " $QM
su $MQUSER -c "/opt/mqm/bin/strmqm $QM"
rc=$?
if [ $rc -ne 0 ]
then
  echo "hamqsi_start_broker_as: Could not start the queue manager"
  exit $rc
fi

# ------------------------------------------------------------------
```

```
# Start the Integration node
#
# Ensure that the integration node is not already running and start the Integration node
su - $MQUSER -c "/MQHA/bin/hamqsi_start_broker $BROKER"
rc=$?
if [ $rc -ne 0 ]
then
  echo "hamqsi_start_broker_as: Could not start the integration node"
  exit $rc
fi

exit $rc
```

4. To stop an integration node, you can use one of the following script files:

   hamqsi_stop_broker

```
#!/bin/ksh
# Module:
#   hamqsi_stop_broker
#
# Args:
#   integration node = name of integration node
#   timeout = max time to allow for each phase of termination
#
# Description:
#   This script stops the integration node, forcibly if necessary.
#   The script should be run by the user account under which
#   the integration node is run, including environment.

BROKER=$1
TIMEOUT=$2

if [ -z "$BROKER" ]
then
  echo "hamqsi_stop_broker: ERROR! No integration node name supplied"
  echo "   Usage: hamqsi_stop_broker <BROKER> <TIMEOUT>"
  exit 1
fi

if [ -z "$TIMEOUT" ]
then
  echo "hamqsi_stop_broker: ERROR! No timeout supplied"
  echo "   Usage: hamqsi_stop_broker <BROKER> <TIMEOUT>"
  exit 1
fi

for severity in normal immediate terminate
do
  # Issue the stop method in the background - we don't
  # want to risk having it hang us up, indefinitely. We
  # want to be able to concurrently run a TIMEOUT timer
  # to give up on the attempt, and try a more forceful
  # stop. If the kill version fails then there is nothing
  # more we can do here anyway.

  echo "hamqsi_stop_broker: Attempting ${severity} stop of ${BROKER}"
  case $severity in

  normal)
    # Minimum severity of stop is to issue mqsistop
    mqsistop $BROKER > /dev/null 2>&1 &
    ;;

  immediate)
    # This is an immediate stop.
    mqsistop $BROKER -i > /dev/null 2>&1 &
    ;;

  terminate)
    # This is a severe method of cleaning up integration node processes.
    # We stop the processes in the following order:
    #   bipservice       - first so it cannot issue restarts
    #   bipbroker        - next for same reason
    #   biphttplistener
    #   bipMQTT
    #   startDataFlowEngine
    #   DataFlowEngine  - last
    for process in bipservice bipbroker biphttplistener startDataFlowEngine DataFlowEngine
    do
      # Output of kill redirected to /dev/null in case no processes
```

```
      ps -ef | grep "$process $BROKER" | grep -v grep | \
        awk '{print $2}'| xargs kill -9 > /dev/null 2>&1
    done
    ;;

  esac

  echo "hamqsi_stop_broker: Waiting for ${severity} stop of ${BROKER} to complete"
  TIMED_OUT=yes
  SECONDS=0
  while (( $SECONDS < ${TIMEOUT} ))
  do
    # See whether there are any integration node processes still running
    cnt=`ps -ef | \
      grep -E "bipservice $BROKER|bipbroker $BROKER|startDataFlowEngine $BROKER|
DataFlowEngine $BROKER|biphttplistener $BROKER" | \
      grep -v grep | wc -l`
    if [ $cnt -gt 0 ]
    then
      # It's still running...wait for timeout
      sleep 1 # loop granularity
    else
      # It's stopped, as desired
      echo "${BROKER} has stopped"
      TIMED_OUT=no
      break # out of while ..offline timeout loop
    fi
  done # timeout loop

  if [ ${TIMED_OUT} = "yes" ]
  then
    continue        # to next level of urgency
  else
    break           # instance is stopped, job is done
  fi

done # next level of urgency

if [ ${TIMED_OUT} = "no" ]
then
  echo "hamqsi_stop_broker: Completed"
  exit 0
else
  echo "hamqsi_stop_broker: Completed with errors"
  exit 1
fi
```

hamqsi_stop_broker_as

```
#!/bin/ksh
# Module:
#   hamqsi_stop_broker_as
#
# Arguments are:
#   integration node = name of integration node
#   qm     = name of integration node queue manager
#   mquser = user account under which QM and integration node run
#   timeout = max time to allow each phase of stop processing
#
# Description:
#   This script stops the integration node, Queue Manager in that sequence.
#
#   Integration node:
#   The script invokes the hamqsi_stop_broker script to stop the
#   integration node, which checks that the integration node is fully stopped.
#
#   Queue Manager:
#   This script uses the strmqm script supplied by MQSeries V7
#
#   The hamqsi_stop_broker_as script should be run as root.


# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi
```

```
BROKER=$1
QM=$2
MQUSER=$3
TIMEOUT=$4

# Check all parameters

if [ -z "$BROKER" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No Integration node name supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No queue manager name supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_stop_broker_as: ERROR! No userid supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
  exit 1
fi

if [ -z "$TIMEOUT" ]
then
  echo "hamsi_stop_broker_as: ERROR! No Timeout value supplied"
  echo "   Usage: hamqsi_stop_broker_as <BROKER> <QM> <MQUSER> <TIMEOUT>"
 exit 1
fi


METHOD_STATUS="OK"

# ------------------------------------------------------------------
# Stop the integration node
#
echo "hamqsi_stop_broker_as: Stop integration node " $BROKER
su - $MQUSER -c "/MQHA/bin/hamqsi_stop_broker $BROKER $TIMEOUT"
if [ $? -ne "0" ]
then
  # Even if the above operation failed, just report and then continue by
  # stopping other components
  echo "hamqsi_stop_broker_as: Attempt to stop integration node $BROKER failed"
  METHOD_STATUS="Error"
fi

# ------------------------------------------------------------------
# Stop the Queue Manager, using script from MQ V7
#
echo "hamqsi_stop_broker_as: Stop Queue Manager $QM"
su $MQUSER -c "/opt/mqm/bin/endmqm -i $QM"
if [ $? -ne "0" ]
then
  # Even if the above operation failed, just report and then continue by
  # stopping other components
  echo "hamqsi_stop_broker_as: Attempt to stop queue manager $QM failed"
  METHOD_STATUS="Error"
fi

if [ ${METHOD_STATUS} = "OK" ]
then
  exit 0
else
  echo "hamqsi_stop_broker_as: Completed with errors"
  exit 1
fi
```

5. To monitor an integration node, you can use the following script file that checks only for the existence of the main integration node processes and provides a successful return code if they are found:

   hamqsi_monitor_broker_as

```
#!/bin/ksh
# Module:
#   hamqsi_monitor_broker_as
```

```
#
# Args:
#   BROKER = name of integration node in AppServer
#   QM     = name of queue manager in AppServer
#   MQUSER = userid under which queue manager and integration node run
#
# Description:
#   This is the application monitor script used with HACMP/ES. It
#   needs to be invoked by a parameter-less wrapper script because
#   HACMP does not allow parameters to be passed to application
#   monitor scripts.
#
#   This hamqsi_monitor_broker_as script is run as root, and uses
#   su as needed to monitor the 3 components of the application server.
#
#   This script is tolerant of a queue manager that is still in
#   startup. If the queue manager is still starting this application
#   monitor script will exit with 0 - which indicates
#   to HACMP that there's nothing wrong. This is to allow for
#   startup time for the queue manager which might exceed the
#   Stabilisation Interval set for the Application Monitor in HACMP/ES.
#
#
# Exit codes:
#   0  => Integration node & QM are all running OK or starting
#   >0 => One or more components are not responding.
#

# Check running as root
if [ `id -u`  -ne 0 ]
then
  echo "Must be running as root"
  exit 1
fi

BROKER=$1
QM=$2
MQUSER=$3

# Check the parameters

if [ -z "$BROKER" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No integration node name supplied"
  exit 1
fi

if [ -z "$QM" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No queue manager name supplied"
  exit 1
fi

if [ -z "$MQUSER" ]
then
  echo "hamqsi_monitor_broker_as: ERROR! No mquser supplied"
  exit 1
fi

# Use a state variable to reflect the state of components as they
# are tested. Valid values are "stopped", "starting" and "started"
# Initialise it to "stopped" for safety.
STATE="stopped"

# ------------------------------------------------------------------
# Check that the queue manager is running or starting.
#
su - $MQUSER -c "echo 'ping qmgr' | runmqsc ${QM}" > /dev/null 2>&1
pingresult=$?
# pingresult will be 0 on success; non-zero on error (man runmqsc)
if [ $pingresult -eq 0 ]
then
  # ping succeeded
  echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is responsive"
  STATE="started"
else
  # ping failed
  # Don't condemn the QM immediately, it might be in startup.
  # The following regexp includes a space and a tab, so use tab-friendly
  # editors.
  srchstr=" $QM[  ]*$"
  cnt=`ps -ef | grep strmqm | grep "$srchstr" | grep -v grep \
```

```
                | awk '{print $2}' | wc -l`
  if [ $cnt -gt 0 ]
  then
    # It appears that QM is still starting up, tolerate
    echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is starting"
    STATE="starting"
  else
    # There is no sign of QM start process
    echo "hamqsi_monitor_broker_as: Queue Manager ${QM} is not responsive"
    STATE="stopped"
  fi
fi


# Decide whether to continue or to exit
case $STATE in
  stopped)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is not running correctly"
    exit 1
    ;;
  starting)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is starting"
    echo "hamqsi_monitor_broker_as: WARNING - Stabilisation Interval might be too short"
    echo "hamqsi_monitor_broker_as: WARNING - No test of integration node $BROKER will be
 conducted"
    exit 0
    ;;
  started)
    echo "hamqsi_monitor_broker_as: Queue manager ($QM) is running"
    continue
    ;;
esac

# ------------------------------------------------------------------
# Check the MQSI integration node is running
#
# Re-initialise STATE for safety
STATE="stopped"
#
# The integration node runs as a process called bipservice which is responsible
# for starting and re-starting the admin agent process (bipbroker).
# The bipbroker is responsible for starting any DataFlowEngines. The
# bipbroker starts the DataFlowEngines using the wrapper script
# startDataFlowEngine. If no integration servers have been assigned to
# the integration node there will be no DataFlowEngine processes. There should
# always be a bipservice and bipbroker process pair. This monitor
# script only tests for bipservice, because bipservice should restart
# bipbroker if necessary - the monitor script should not attempt to
# restart bipbroker and it might be premature to report an absence
# of a bipbroker as a failure.
#
cnt=`ps -ef | grep "bipservice $BROKER" | grep -v grep | wc -l`
if [ $cnt -eq 0 ]
then
  echo "hamqsi_monitor_broker_as: MQSI integration node $BROKER is not running"
  STATE="stopped"
else
  echo "hamqsi_monitor_broker_as: MQSI integration node $BROKER is running"
  STATE="started"
fi

# Decide how to exit
case $STATE in
  stopped)
    echo "hamqsi_monitor_broker_as: Integration node ($BROKER) is not running correctly"
    exit 1
    ;;
  started)
    echo "hamqsi_monitor_broker_as: Integration node ($BROKER) is running"
    exit 0
   ;;
esac
```

If you require more information than that supplied by the preceding example, you can code your monitor to do the following actions:

- Subscribe to IBM App Connect Enterprise accounting and statistics, and analyze the results.
- Put a dummy message through the integration node and analyze the results.

6. Before you delete the integration node on the primary node, you must delete any integration nodes on the standby nodes. For more information, see “Deleting an integration node” on page 165.

Using an integration node with an existing Windows Cluster (Windows Server)

You can use IBM App Connect Enterprise with the existing high availability manager for Windows Server (Failover Cluster Manager).

About this task

For more information about the versions of Windows Server that are supported by IBM App Connect Enterprise, see the IBM App Connect Enterprise system requirements web page.

This topic summarizes how to complete the following tasks:

1. Complete the prerequisite setup.
2. Configure a local group.
3. Create an integration node.
4. Add integration node instances to the additional nodes.
5. Add the integration node service to the cluster configuration
6. Start and stop an integration node.
7. Delete an integration node.

Procedure

1. To complete the prerequisite setup, complete the following steps:
   a) Validate your Failover Cluster Manager configuration.
   b) Complete all of the steps documented in 'Supporting the Microsoft Cluster Service' of the IBM MQ documentation to create a cluster configuration that contains the queue manager on which you want the integration node to run.
   c) Note the `haregtyp.exe` command as a prerequisite for creating IBM MQ resources in the cluster.
2. To configure a local group, ensure that the domain user under which you want the integration node to run exists in the local `mqbrkrs` group on all nodes on which you want the integration node to run.
3. To create an integration node, use the following **mqsicreatebroker** command on the node on which your cluster is currently running.

   ```
   mqsicreatebroker INODE -q MQ1 -e E:\ACE\Workspace
   ```

   where:

   - INODE is the name of the integration node.
   - MQ1 is the name of the queue manager.
   - E:\ACE\Workspace is the directory of a shared disk (nonquorum) in your cluster configuration.
4. To add this integration node instance to the other nodes in your cluster, switch your cluster to each node in turn. When a node is active, use the following **mqsiaddbrokerinstance** command.

   ```
   mqsiaddbrokerinstance INODE -e E:\ACE\Workspace
   ```

   where:

   - INODE is the name of the integration node.
   - E:\ACE\Workspace is the directory of a shared disk (nonquorum) in your cluster configuration.

5. To add an integration node generic service resource to the cluster which contains the integration node queue manager, complete the following steps:
   a) Select the IBM App Connect Enterprise component `integration_node` service when asked. All other settings can be left unchanged.
   b) Add a dependency on the IBM MQ resource, to ensure that the queue manager is started before the integration node.
6. To start and stop the integration node resource, use Failover Cluster Manager.
7. To delete an integration node resource from the Failover Cluster Manager, take the following steps:
   a) Delete the integration node generic service from the cluster configuration.
   b) Use the **mqsiremovebrokerinstance** command on all nodes but one, moving the cluster between nodes before running each command.
   c) On the final node, use the **mqsideletebroker** command to completely remove the integration node configuration.

Results

For further information about configuring IBM MQ with a high availability manager for Windows Server, see the IBM MQ product documentation online.

See the following topics:

- Introducing MSCS clusters
- Supporting the Microsoft Cluster Service (MSCS)
- Setting up IBM MQ for MSCS clustering

Using an integration node with an RDQM configuration

You can use IBM App Connect Enterprise with an IBM MQ replicated data queue manager (RDQM), to provide a high availability (HA) solution.

About this task

An RDQM configuration consists of three servers configured in a high availability (HA) group, each with an instance of the queue manager. One instance is the running queue manager, which synchronously replicates its data to the other two instances. If the server running this queue manager fails, another instance of the queue manager starts and has current data to work with. The three instances of the queue manager share a floating IP address, so clients need to be configured with only a single IP address. Only one instance of the queue manager can run at any one time, even if the HA group becomes partitioned due to network problems. The server running the queue manager is known as the primary, and each of the other two servers is known as a secondary.

For more information about RDQM, see the IBM MQ product documentation online.

You can configure an integration node to work with an RDQM high availability solution, by completing the following steps:

Procedure

1. Install IBM App Connect Enterprise on all hosts that are part of the RDQM cluster.
2. On all hosts, add the `mqm` user ID and your login user ID to the **mqbrkrs** group.
3. Restart the queue manager to ensure that it gains the **mqbrkrs** group membership.
4. Create a directory in the queue manager `vols` directory for the integration node, as shown in the following example:

```
# mkdir /var/mqm/vols/ha1/userdata/ace
# chown mqm:mqbrkrs /var/mqm/vols/ha1/userdata/ace
```

```
# chmod 775 /var/mqm/vols/ha1/userdata/ace
```

5. Use the **mqsicreatebroker** command to create an integration node called ACEHA1, set the new data directory, and specify that the integration node is to be started and stopped as an MQ service when the queue manager starts and stops:

```
# mqsicreatebroker -e /var/mqm/vols/ha1/userdata/ace -d defined -q HA1 ACE1HA1
```

6. Configure the integration node by editing its `node.conf.yaml` configuration file:

```
# vi /var/mqm/vols/ha1/userdata/ace/mqsi/components/ACEHA1/node.conf.yaml
```

   If your system has a limited amount of memory, consider reducing the amount of memory required by IBM App Connect Enterprise by setting the **startListener** property in the **NodeHttpListener** section to `false` to turn off the biphttplistener process:

```
NodeHttpListener:
  startListener: false
```

7. Start the integration node by running the **mqsistart** command:

```
# mqsistart ACEHA1
```

8. Verify that the integration node is running by using the **mqsilist** command:

```
 # mqsilist
```

   A response similar to the following message confirms that the integration node is running:

```
BIP1376I: Integration node 'ACEHA1' is an active multi-instance or High Availability
 integration node that is running on queue manager 'HA1'.
The administration URI is 'http://admin.myloc.company.com:4414'
BIP8071I: Successful command completion.
```

   If the integration node does not start, check for FDC files in the `/var/mqm/errors` directory and the contents on the queue manager error log. Also check the contents of the service log file:

```
/var/mqsi/common/log/ACEHA1.mqservice.log
```

9. Create an integration server on integration node ACEHA1:

```
# mqsicreateexecutiongroup -e default ACEHA1
```

10. Create the instances on the other nodes by using the **mqsiaddbrokerinstance** command. This command cannot be run unless the `/var/mqm/vols/ha1` filesystem exists, so you must fail over to each node in turn by using the IBM MQ **rdqmadm** command and then add the broker instance by using the **mqsiaddbrokerinstance** command.

    For each node, complete the following steps:

    a) Run the IBM MQ command **rdqmadm**, specifying the name of the queue manager (for example, HA1):

```
# rdqmadm -p -m HA1
```

    b) Run the IBM App Connect Enterprise command **mqsiaddbrokerinstance**, specifying the integration node name (for example, ACEHA1) and the filesystem to be used:

```
# mqsiaddbrokerinstance ACEHA1 -e /var/mqm/vols/ha1/userdata/ace
```

11. You have now configured an RDQM solution.

HTTP proxy servlet overview

By using the HTTP proxy servlet in a servlet container, you can support high availability, load distribution, access to the integration node from multiple IP addresses and ports, and a larger number of concurrent HTTP sessions.

The HTTP proxy servlet is a Java servlet that you can use in a servlet container, such as WebSphere Application Server Liberty Profile or Apache Tomcat, or in an application server such as IBM WebSphere Application Server. The HTTP proxy servlet receives HTTP requests from web services client applications and replaces the support that is provided by the integration node and embedded (integration server) HTTP listeners.

The HTTP proxy servlet supports SSL (HTTPS) secure protocol when it is deployed in a servlet container that is configured to support SSL.

If you are deploying a web browser-based JavaScript application that uses the JavaScript client API to call an IBM App Connect Enterprise integration service, then you must deploy the proxy servlet on the same web application server as your JavaScript application. For more information about the JavaScript client API, see “Integration service JavaScript client API ” on page 1995.

You cannot use the HTTP proxy servlet if you configure your integration node environment to use multi-instance IBM MQ queue managers, because the HTTP proxy servlet cannot connect to the standby queue manager when it becomes active.

For a detailed description of the HTTP proxy servlet, see “HTTP proxy servlet operation” on page 141.

Before you install and test the HTTP proxy servlet, ensure that you understand the following concepts:

- “Working with HTTP flows” on page 792
- “HTTP listeners” on page 795
- “HTTP proxy servlet operation” on page 141
- “Importance of the context root when you configure the HTTP proxy servlet” on page 147

When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, install, and test the HTTP proxy servlet:

- “Configuring the HTTP proxy servlet” on page 149
- “Deploying the HTTP proxy servlet on a servlet container” on page 159
- “Testing the HTTP proxy servlet” on page 164

HTTP proxy servlet operation

Before you deploy the HTTP proxy servlet, ensure that you understand how the HTTP proxy servlet manages the communication between IBM App Connect Enterprise applications and web services clients in a number of supported configurations.

The HTTP proxy servlet is a Java Web Application Archive (`.war`) file that is part of the runtime environment, and can be found in the following directory:

install_dir`/tools`

where *install_dir* is the name of your App Connect Enterprise runtime installation directory.

The HTTP proxy servlet is a Java servlet that receives HTTP requests. The HTTP proxy servlet matches the received web address with the web address that the HTTP or SOAP input nodes are monitoring, then passes the HTTP request message to the correct HTTP or SOAP input node flow by using IBM MQ.

The HTTP proxy servlet receives response messages from the HTTP or SOAP reply nodes and sends them back to the client applications over HTTP or HTTPS. The integration node has several internal IBM MQ queues (SYSTEM.BROKER.WS.* queues), which are used for the communication between the HTTP proxy servlet and the HTTP or SOAP input and reply nodes.

The HTTP proxy servlet is deployed in a *servlet container*. A servlet container (or web container) is the runtime environment for servlets and Java Server Pages (JSP). Apache Tomcat is an example of a servlet container. WebSphere Application Server is an example of an application server, which includes the

functions of a servlet container. The HTTP proxy servlet can be deployed in a local servlet container that is running on the same machine as IBM App Connect Enterprise or on a remote servlet container that is running on a different machine to IBM App Connect Enterprise. The servlet container must allow the HTTP proxy servlet to configure and call the IBM MQ classes for Java.

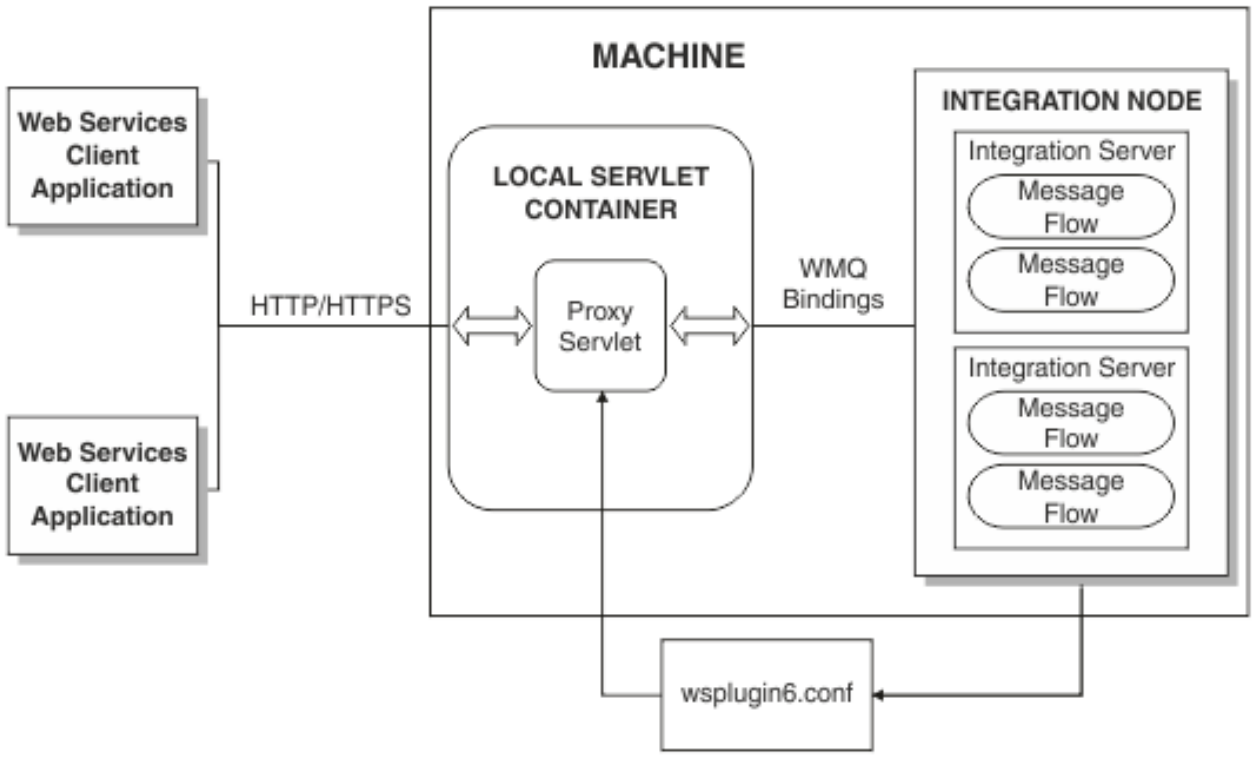
After the HTTP proxy servlet is deployed and running on the servlet container, it uses the HTTP listener of the servlet container to receive HTTP requests. If the servlet container is configured to support SSL (HTTPS), web services requests are received by the message flows by using SSL.

The following topics describe some common scenarios that are based on typical IBM App Connect Enterprise deployments, and the HTTP proxy servlet configuration parameters that are required:

- The HTTP proxy servlet is on the same machine as IBM App Connect Enterprise
- The HTTP proxy servlet is on a different machine to IBM App Connect Enterprise
- The HTTP proxy servlet is on a machine with its own queue manager
- The HTTP proxy servlet is deployed in an environment with a network local-balancer

HTTP proxy servlet is deployed to a servlet container on the same machine as IBM App Connect Enterprise
You can configure the HTTP proxy servlet for a deployment where the HTTP proxy servlet is deployed to a servlet container on the same machine as IBM App Connect Enterprise.

In the following figure, the HTTP proxy servlet is running on the same machine as the integration node. The HTTP proxy servlet connects to the integration node queue manager by using bindings mode (local connection).



In this configuration example, you must configure the following HTTP proxy servlet configuration parameters. For information about configuring the HTTP proxy servlet, see “Configuring the HTTP proxy servlet” on page 149.

- Set **useClientMode** to `false`.
- Set **useQueueManagerDataInsteadOfConfigFile** to "".
- Set **configFile** to the location of the configuration file, for example, `C:\ProgramData\IBM\MQSI\components\`*`my_node`*`\config\wsplugin6.conf`.

where *my_node* is the name of your integration node.

Before you configure, deploy, and test the HTTP proxy servlet, ensure that you understand the following concepts:

- “Working with HTTP flows” on page 792
- “HTTP listeners” on page 795
- “HTTP proxy servlet operation” on page 141
- “Importance of the context root when you configure the HTTP proxy servlet” on page 147

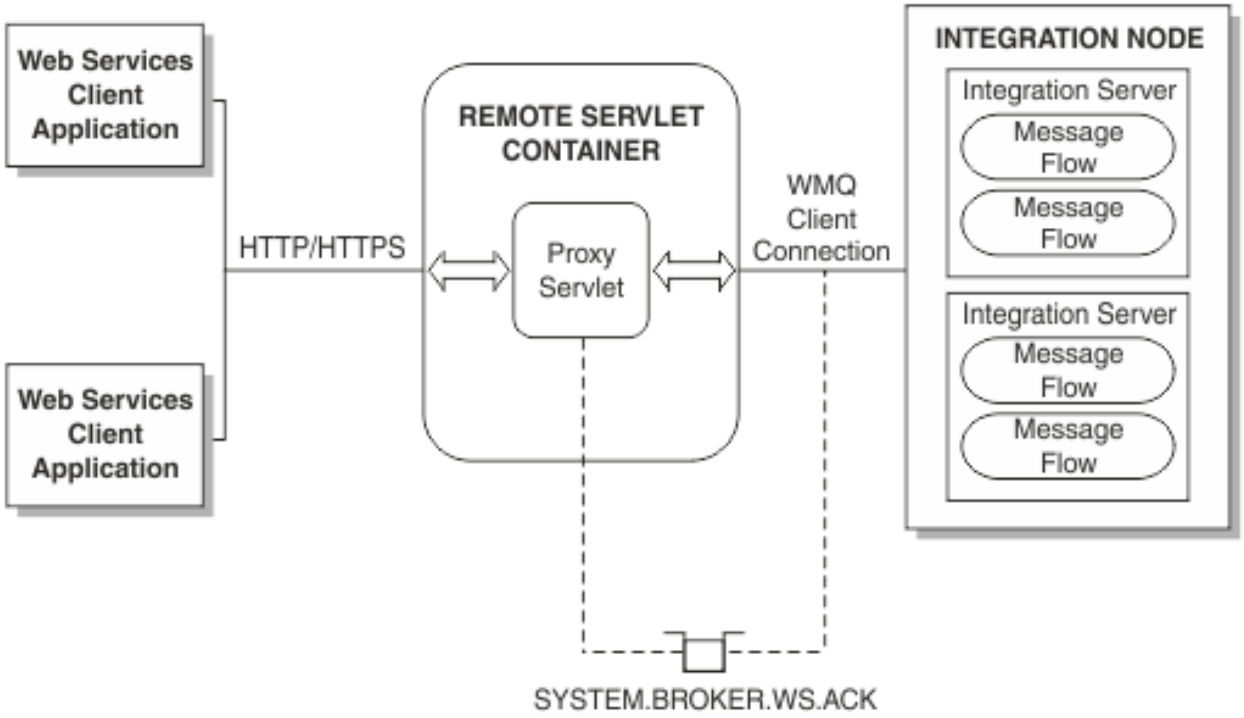
When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, deploy, and test the HTTP proxy servlet:

- “Configuring the HTTP proxy servlet” on page 149
- “Deploying the HTTP proxy servlet on a servlet container” on page 159
- “Testing the HTTP proxy servlet” on page 164

HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise

You can configure the HTTP proxy servlet for a deployment where the HTTP proxy servlet is deployed to a servlet container on a different machine to IBM App Connect Enterprise, with an IBM MQ client link to the integration node queue manager.

In the following figure, the HTTP proxy servlet is running on a remote machine to the integration node. The HTTP proxy servlet connects to the integration node queue manager by using an IBM MQ client connection, the HTTP proxy servlet can be configured to access HTTP or SOAP nodes, and the HTTP or SOAP node configuration is retrieved from the SYSTEM.BROKER.WS.ACK queue.



In this configuration example, you must configure the following HTTP proxy servlet configuration parameters. For information about configuring the HTTP proxy servlet, see “Configuring the HTTP proxy servlet” on page 149.

- Set **useClientMode** to `true`.

- Set **useQueueManagerDataInsteadOfConfigFile** to * or the integration node queue manager name.
- Set **clientModeHostname**, **clientModeChannelName**, and **clientModePortNumber** to the correct values as detailed in "HTTP proxy servlet configuration parameters" on page 151.

The HTTP proxy servlet attempts to connect to the remote queue manager for the integration node by reading the required configuration data from the IBM MQ client connection. For the client connection to succeed, you must start the SYSTEM.DEFAULT.LISTENER.TCP listener in the queue manager. For more information, search for *START LISTENER* in the IBM MQ product documentation.

Before you configure, deploy, and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "Working with HTTP flows" on page 792
- "HTTP listeners" on page 795
- "HTTP proxy servlet operation" on page 141
- "Importance of the context root when you configure the HTTP proxy servlet" on page 147

When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, deploy, and test the HTTP proxy servlet:

- "Configuring the HTTP proxy servlet" on page 149
- "Deploying the HTTP proxy servlet on a servlet container" on page 159
- "Testing the HTTP proxy servlet" on page 164

HTTP proxy servlet is deployed to a servlet container on a machine with its own queue manager
You can configure the HTTP proxy servlet for a deployment where the HTTP proxy servlet is deployed to a servlet container that is on a machine with its own queue manager and an IBM MQ channel link to the integration node queue manager.

In this configuration example, you must configure the following HTTP proxy servlet configuration parameters. For information about configuring the HTTP proxy servlet, see "Configuring the HTTP proxy servlet" on page 149.

- Set **useClusterMode** to `true`.
- Set **clusterModeQueueManagerName** to the queue manager of the Web servlet container.
- Set **clusterModeReplyToQ** to a queue that exists on that queue manager.

The HTTP proxy servlet attempts to open the queue SYSTEM.BROKER.WS.INPUT on the specified queue manager by using the queue manager name from the configuration file. Therefore, you must set up channels and transmit queues beforehand to ensure that the messages arrive on the queue manager of the integration node.

You must copy the configuration files from the integration node machine in this scenario.

Before you configure, deploy, and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "Working with HTTP flows" on page 792
- "HTTP listeners" on page 795
- "HTTP proxy servlet operation" on page 141
- "Importance of the context root when you configure the HTTP proxy servlet" on page 147

When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, deploy, and test the HTTP proxy servlet:

- "Configuring the HTTP proxy servlet" on page 149
- "Deploying the HTTP proxy servlet on a servlet container" on page 159
- "Testing the HTTP proxy servlet" on page 164

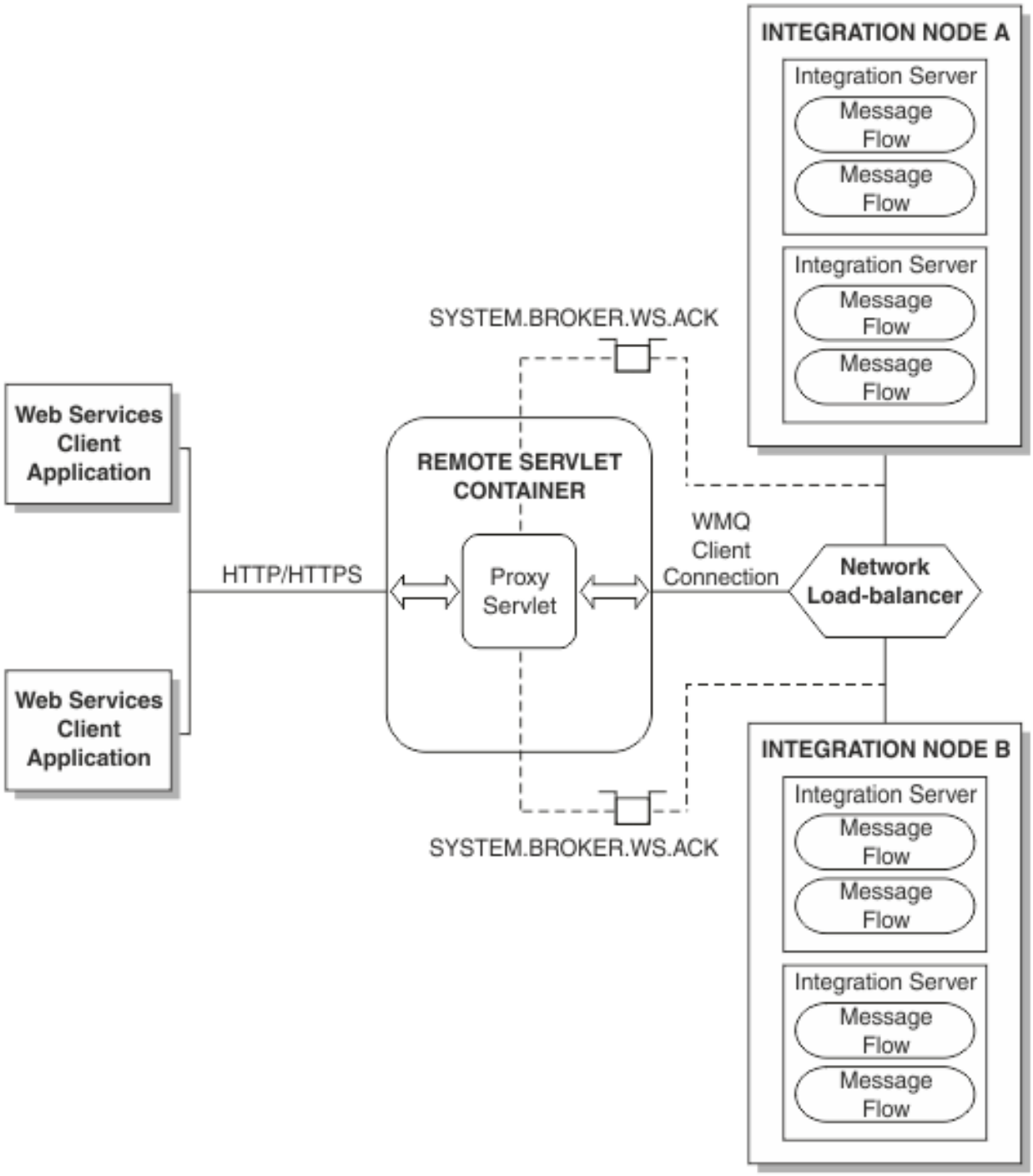
HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise with a network load-balancer for distributing work to several integration nodes

You can configure the HTTP proxy servlet for a deployment where the HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise, and the servlet container has an IBM MQ client link to the integration node queue manager. In this configuration, a network load-balancer is used to distribute work to several integration nodes.

In the following figure the HTTP proxy servlet is configured to load balance IBM MQ connections across multiple integration nodes. A network load balancer is required for this configuration.

When the HTTP proxy servlet is configured to connect to multiple integration nodes, the integration nodes must be identical clones of each other, which means that the same HTTP and SOAP flows are deployed with the same web addresses.

The HTTP proxy servlet sends the HTTP requests over the IBM MQ connections to distribute the load between the active integration node connections.



The configuration is the same as that described in "HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise" on page 143, but the network load-balancer replaces the integration node machine. Configuration files cannot be used because there are several integration nodes behind one virtual IP address, and each one has a different configuration file. The HTTP proxy servlet loads information for each connection, and uses the relevant configuration information for each integration node.

In this configuration example, you must configure the following HTTP proxy servlet configuration parameters. For information about configuring the HTTP proxy servlet, see "Configuring the HTTP proxy servlet" on page 149.

- Set **useClientMode** to `true`.

- Set **useQueueManagerDataInsteadOfConfigFile** to * or the integration node queue manager name.
- Set **clientModeHostname**, **clientModeChannelName**, and **clientModePortNumber** to the correct values as detailed in "HTTP proxy servlet configuration parameters" on page 151.

If failover is one of the reasons for deploying this configuration, it is recommended that you configure the following additional HTTP proxy servlet configuration parameters:

- Set **clientModeConnectRetryCount** to a value equal or higher than the number of integration nodes. This setting ensures that a single failed server does not cause intermittent errors, even if the load-balancer does simple round-robin scheduling. The HTTP proxy servlet uses the first available integration node.
- Set **reconnectActiveLinksAge** to a value less than the firewall timeout. This setting prevents the reuse of old connections that might have been discarded by firewalls between the HTTP proxy servlet and the load-balancer (or between the load-balancer and the integration nodes).

You can set **testConnectionBeforeReuse** to `true` as an alternative way to manage dropped IBM MQ connections between the HTTP proxy servlet and integration node queue managers. However, this option causes an MQINQ call to be performed before any attempt to send data to the integration node. If the MQINQ call fails, a new connection is established, and the data is sent over the new connection. Because the configuration adds another operation to the MQPUT and MQGET calls, it results in significant processing for every message; use this option only if no alternative options are available. For information about MQINQ, MQPUT, and MQGET calls, search for *Function calls* in the IBM MQ product documentation.

Before you configure, deploy, and test the HTTP proxy servlet, ensure that you understand the following concepts:

- "Working with HTTP flows" on page 792
- "HTTP listeners" on page 795
- "HTTP proxy servlet operation" on page 141
- "Importance of the context root when you configure the HTTP proxy servlet" on page 147

When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, deploy, and test the HTTP proxy servlet:

- "Configuring the HTTP proxy servlet" on page 149
- "Deploying the HTTP proxy servlet on a servlet container" on page 159
- "Testing the HTTP proxy servlet" on page 164

Importance of the context root when you configure the HTTP proxy servlet

You must assign a context root to the HTTP proxy servlet when you deploy the HTTP proxy servlet to a servlet container. The choice of context root determines the message flow nodes or integration service URLs with which the HTTP proxy servlet can communicate.

Web addresses, or Universal Resource Locators (URLs), have an important role when HTTP or SSL (HTTPS) protocols are used.

The HTTP proxy servlet passes the requests from the servlet container to the integration node and vice versa.

Each HTTP or SOAP input node expects to receive requests from a specific web address (or web addresses when wildcard characters are used). The servlet container also uses the web address to locate the servlets that are going to process the HTTP or HTTPS requests that the servlet container receives on behalf of the integration node.

The web address is structured in the following way:

```
schema://host_name:port/url_path
```

where:

schema
Specifies the protocol (HTTP or HTTPS).

host_name
Specifies the host name or IP address of the server where the servlet container is running.

port
Specifies the port number on which the servlet container is listening.

url_path
Specifies the URL path; a series of tokens (separated by slashes /) that identify both the HTTP proxy servlet, and the HTTP or SOAP input nodes.

When you use the HTTP proxy servlet, the URL path is partitioned as follows:

```
context_root/node_url_path
```

where:

context_root
Specifies the part of the URL path (context root) that is allocated to the proxy servlet by the servlet container when the proxy servlet is deployed.

node_url_path
Specifies the part of the URL path that makes the web address unique to a specific HTTP or SOAP input node.

Note: You can set *context_root* to / in which case *url_path* and *node_url_path* are the same.

The entire URL path must be configured in the properties of the HTTPInput or SOAPInput node. For more information, see node and node.

For example, you might have HTTPInput nodes that are configured to receive requests for the following web addresses:

- Node1: `http://myhost.com/app1`
- Node2: `http://myhost.com/public/app2`
- Node3: `http://myhost.com/public/app3`
- Node4: `http://myhost.com/private/app4`

If you set the context root for your HTTP proxy servlet to `/public`, then you can communicate with Node2 and Node3 via the HTTP proxy servlet.

If you set the context root for your HTTP proxy servlet to `/private`, then you can communicate with Node4 via the HTTP proxy servlet.

If you set the context root for your HTTP proxy servlet to `/`, then you can communicate with all of the HTTPInput nodes via the HTTP proxy servlet.

Choosing a context root for the HTTP proxy servlet

- If you have many existing applications that you want to access with the HTTP proxy servlet, and the input nodes or integration services do not use URLs with a common root, you might want to set the context root for the HTTP proxy servlet to `/`. Then you do not need to modify the URLs used by your input nodes or integration services. However, if you set the context root for the HTTP proxy servlet to `/`, then the HTTP proxy servlet becomes the default application for the servlet container, and the HTTP proxy servlet attempts to process all requests that do not match the context root of any other applications that are hosted by the servlet container.
- If you want to access a subset of your existing applications with the HTTP proxy servlet, you might configure the URLs for the input nodes or integration services to start with a consistent value, for example `/public`. Then you must set the context root of the HTTP proxy servlet to the same value.
- If you are building applications from scratch, then consider the requirements of the HTTP proxy servlet when you plan the structure of the URLs you configure for your input nodes or integration services.

Before you install and test the HTTP proxy servlet, ensure that you understand the following concepts:

- “Working with HTTP flows” on page 792
- “HTTP listeners” on page 795
- “HTTP proxy servlet operation” on page 141

When you have gained an understanding of the HTTP proxy servlet concepts, read the following topics to help you configure, install, and test the HTTP proxy servlet:

- “Configuring the HTTP proxy servlet” on page 149
- “Deploying the HTTP proxy servlet on a servlet container” on page 159
- “Testing the HTTP proxy servlet” on page 164

Configuring the HTTP proxy servlet

Configure the HTTP proxy servlet with the details of the integration node environment to which the HTTP proxy servlet connects. The HTTP proxy servlet must be configured before you deploy the HTTP proxy servlet to a servlet container.

Before you begin

In order that the HTTP proxy servlet can access the SOAPInput and SOAPReply nodes, you must enable the integration node listener for each integration server where message flows with SOAP nodes are deployed. See “HTTP listeners” on page 795 and “Switching from embedded listeners to an integration node listener” on page 798.

About this task

The HTTP proxy servlet `proxyservlet.war` file is part of the runtime environment, and can be found in the following directory:

install_dir`/server/tools`

where *install_dir* specifies the name of your runtime installation directory.

Complete the following steps to configure the web deployment descriptor (`web.xml`) for the HTTP proxy servlet by using the IBM App Connect Enterprise Toolkit.

Procedure

1. To import and configure a `.war` file, you must have the Eclipse Java EE developer Tools feature installed in the IBM App Connect Enterprise Toolkit. This feature is not included in the IBM App Connect Enterprise Toolkit by default but can be installed by completing the following instructions:
   a) From the IBM App Connect Enterprise Toolkit menu, click **Help** > **Install New Software**.
   b) Enter the following URL in the "Work with" field:
      `http://download.eclipse.org/releases/juno`
   c) Click **Add** and enter a name for the site.
      For example, `Eclipse Juno downloads`.
   d) Click **OK** and wait until the available features are listed.
   e) Expand the **Web, XML, Java EE and OSGi Enterprise Development** category, select the **Eclipse Java EE Developer Tools** feature and click **Next**.
      The **Install Details** dialog is displayed.
   f) Click **Next**, accept the license, and then click **Finish**.
      The feature is installed.
   g) Click **Yes** when you are prompted to restart the IBM App Connect Enterprise Toolkit.

   The **Eclipse Java EE Developer Tools** feature is now available in the IBM App Connect Enterprise Toolkit.
2. From the IBM App Connect Enterprise Toolkit menu, switch to the Java EE perspective by clicking **Window** > **Open Perspective** > **Other** and clicking **Java EE**.
3. Click **File** > **Import**, expand the **Web** section, select **WAR file** in the list, and click **Next**.
4. Click **Browse** to find the `proxyservlet.war` file in *`install_dir`*`\server\tools`, where *install_dir* specifies the name of your IBM App Connect Enterprise installation directory (for example, `C:\Program Files\IBM\ACE\12.0.`*`n`*`.0\server\tools\proxyservlet.war`), and click **Open**.
5. Set the name of the Web project to `proxyservlet` and click **Finish**.
   The HTTP proxy servlet is now ready for configuring by using the Java EE perspective.
6. In the **Project Explorer** view, expand **proxyservlet** and double-click **Deployment Descriptor** to view the web deployment descriptor.
7. Find the **Servlets and JSPs** section in the Web Deployment Descriptor, and click the servlet link called **WBIMBServlet** to display the servlet web address mappings and initialization parameters.
   The same parameters in the `web.xml` file can be configured through JNDI in WebSphere Application Server. This alternative method means that you set up at the application server side only once for any future deployment of the proxy servlet. This operation is possible because the JNDI configuration parameters take precedence over the initialization parameters in the `web.xml` file. For more information about setting up the JNDI interface for the proxy servlet, see "Setting up the JNDI interface for the proxy servlet" on page 156.
8. Click the **Source** tab, which is found at the bottom of the **Deployment Descriptor** view.
   The source of the web deployment descriptor (`web.xml`) displays the HTTP proxy servlet parameters.
9. Edit the HTTP proxy servlet parameters as required; see "HTTP proxy servlet configuration parameters" on page 151.
10. When the configuration is complete, save the changes to the `web.xml` file by pressing **Ctrl+S**.

11. Export the configured HTTP proxy servlet ready for deployment to a servlet container by completing the following steps:
    a) In the **Project Explorer** view, right-click the **Deployment Descriptor** for the proxyservlet Web project, and click **Export** > **WAR file**.

       The **WAR Export** dialog window is displayed.
    b) Click **Browse** and specify a location and a name for the configured WAR file. For example, `myproxyserlet.war`.
    c) Click **Save** and then click **Finish**.

Results

You have now configured the HTTP proxy servlet with the initialization parameters.

What to do next

Deploy and configure the HTTP proxy servlet on the web application server that will host the JavaScript application, see “Deploying the HTTP proxy servlet on a servlet container” on page 159.

HTTP proxy servlet configuration parameters
Before you can deploy the HTTP proxy servlet to the servlet container, you must configure it with the following initialization parameters for the integration node environment to which the HTTP proxy servlet connects.

The parameters that you can configure for the HTTP proxy servlet are detailed in the following sections. For information about how to configure the HTTP proxy servlet, see “Configuring the HTTP proxy servlet” on page 149.

- “General options” on page 152
- “Information options” on page 153
- “ReplyToQ and QMgr options” on page 153
- “SSL connection options” on page 153
- “MQ connection options” on page 154

The following topics describe some common scenarios and the HTTP proxy servlet configuration parameters that are required.

- “HTTP proxy servlet is deployed to a servlet container on the same machine as IBM App Connect Enterprise” on page 142
- “HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise” on page 143
- “HTTP proxy servlet is deployed to a servlet container on a machine with its own queue manager” on page 144
- “HTTP proxy servlet is deployed to a servlet container that is on a different machine to IBM App Connect Enterprise with a network load-balancer for distributing work to several integration nodes” on page 145

General options

Parameter name	Default value	Description
brokerName	*	Set to * or the name of your integration node. Use this parameter to specify the name that is used for error messages; the value is auto-detected if set to "*". Specify a value if several integration nodes are being used by the HTTP proxy servlet, and a single name is required for error messages.
configFilePath	`/var/mqsi/components/INODE/config/wsplugin6.conf`	Specify the full path of the configuration file. If the integration node that is used by the HTTP proxy servlet is local, set this parameter to the `wsplugin6.conf` file. This value is used only when the parameter **useQueueManagerDataInsteadOfConfigFile** is set to blank. The configuration file is used only when the HTTP proxy servlet is running on the same server as the integration node, and it has access to the file. In Windows, the file is stored in `C:\`*`install_dir`*`\config\wsplugin.conf` or `C:\Documents and Settings\All Users\IBM\MQSI\components\`*`NodeName`*`\config\wsplugin6.conf` where *NodeName* is the name of your integration node. On Linux and UNIX, the file is stored in `/var/mqsi/config/wsplugin.conf` or in `/var/mqsi/components/`*`NodeName`*`/config/wsplugin6.conf` where *NodeName* is the name of your integration node.
useFastpathBindingsConnection	false	Specify `true` or `false`. Set the value to `true` to make the HTTP proxy servlet connect in fastpath mode if the HTTP proxy servlet is using a local queue manager.
traceFileName		Specify the full path of the trace file. If this parameter is not specified the trace is sent to stdout.
turnTraceOn	0	Set to `0`, `1`, or `2`. Set `0` for no trace, `1` for normal trace, or `2` for debug trace.

Information options

Parameter name	Default value	Description
enableStatusPage	false	Set to `true` or `false`. Set the value to `true`, to make the page accessible at `http://`*`HostName`*`:`*`Port`*`/`*`ContextRoot`*`/messagebroker/httpproxy/statuspage` where *HostName* is the name of your servlet container, *Port* is the port for your servlet container, and *ContextRoot* is the context root you have set for your HTTP proxy servlet. For more information about the context root, see "Importance of the context root when you configure the HTTP proxy servlet" on page 147.
enableInfoHeaders	false	Set to `true` or `false`. Set the value to `true` to make the HTTP proxy servlet add extra headers in the response message. These headers are: • X-WMB-Broker-Name • X-WMB-QM-Name • X-WMB-MQ-URL-CorrelId The headers also contain details of the configuration that is used for the response message.

ReplyToQ and QMgr options

Parameter name	Default value	Description
useClusterMode	false	Set to `true` or `false`. Set to `true` if the HTTP proxy servlet is required to put reply-to queue and queue manager information in the MQMD of sent messages to enable the integration node to respond to the correct queue manager in a cluster.
clusterModeQueueManagerName	SOME_OTHER_QUEUE_MANAGER	Set to the queue manager name for initial MQCONN and ReplyToQMgr.
clusterModeReplyToQ	YOUR.REPLYTO.QUEUE	Set to the reply queue name on which to listen.

SSL connection options

Parameter name	Default value	Description
useSecuredChannel	false	Set to `true` or `false`. Set to `true` if SSL is configured on the MQ channel. If set to `true`, the proxy servlet attempts to establish a secured connection to the MQ channel by using the `keyStore`, `keyStorePassword`, `trustStore`, `trustStorePassword`, and `cipherSuite` parameter values.

Parameter name	Default value	Description
keyStore		Set to the full path of the keystore file. The fully qualified path to the keystore file, which is of type "JKS". For example, in Windows: `C:\\Program Files\ \IBM\\MQSI\\keystore.jks` On Linux and UNIX: `/var/mqsi/keystore.jks`
keyStorePassword	changeit	Set to the password of the keystore file.
trustStore		Set to the full path of the truststore file. The fully qualified path to the truststore file, which is of type "JKS". For example, on Windows: `C:\\Program Files\ \IBM\\MQSI\\truststore.jks` On Linux and UNIX: `/var/mqsi/truststore.jks` This field is mandatory if `useSecuredChannel` is set to `true`.
trustStorePassword	changeit	Set to the password of the truststore file.
cipherSuite		Set to the encryption type that is configured in the MQ channel. For example: `SSL_RSA_WITH_NULL_MD5` This field is mandatory if `useSecuredChannel` is set to `true`.

MQ connection options

Parameter name	Default value	Description
useClientMode	false	Set to `true` or `false`. Set the value to `true` to use the IBM MQ client or set the value to `false` to use the bindings connection. Normally, **useQueueManagerDataInsteadOfConfigFile** would also be set to the integration node queue manager if this parameter is set to `true`.
clientModeHostname	localhost	Set to the host name or the IP address for the queue manager.
clientModeChannelName	SYSTEM.DEF.SVRCONN	Set to the IBM MQ SVRCONN channel name
clientModePortNumber	1414	Set to the port number of the IBM MQ listener.

Parameter name	Default value	Description
clientModeConnectRetryCount	1	Specify the number of times to retry the IBM MQ connect call. Use this parameter in cases where a network dispatcher or load balancer is used to distribute work to a set of queue managers and one of the queue managers fails. A new connect call might fail the first time, but succeed the second time. The retry count must be set to a high number to provide the greatest chance of success.
useQueueManagerDataInsteadOfConfigFile		Set to the queue manager name, * (remote proxy), or leave blank (local proxy). Set this value to the queue manager name to make the HTTP proxy servlet retrieve web address data from a queue, and avoid the need for a configuration file to be accessible by the HTTP proxy servlet.
sleepBeforeGet	0	Set to the time (in seconds) to wait before the HTTP proxy servlet issues an MQGET call for a response message from the integration node.
disconnectBeforeSleep	true	Set to `true` or `false`. Set the value to `true` to release the IBM MQ handle while sleeping and minimize the number of simultaneous IBM MQ connections.
reconnectActiveLinksAge	-1	Set to a time (in seconds), `0`, or `-1` Set the value to a number greater than zero to force IBM MQ connections to be disconnected and reconnected if they have been inactive, because of low traffic volumes, for more than the specified number of seconds. Set the value to `-1` to prevent any reconnection. Set the value to `0` to force all connections to be used only one time. This parameter is useful if the connection to IBM MQ goes through a firewall that closes connections after a period of inactivity. Set the value to a value less than the firewall timeout to prevent clients from receiving IBM MQ 2009 (connection broken) error code responses.

Parameter name	Default value	Description
testConnectionBeforeReuse	false	Set to `true` or `false`. Set the value to `true`, to make the HTTP proxy servlet attempt an MQINQ before it does the MQPUT of the HTTP data message. All problems with a cached IBM MQ client connection are detected at that point, and a new connection is established for the MQPUT of the actual data (and MQGET of the response). This parameter causes significant extra network traffic, and must be used only if there are problems with dropped connections, which are usually seen as IBM MQ 2009 errors, indicating that the connection is broken.
maximumConnectionAge	-1	Set to a time (in seconds), `0`, or `-1` Set the value to a number greater than zero to disconnect and reconnect IBM MQ connections if they are older than the specified number of seconds. Set the value to `-1` to prevents any reconnections. Set the value to `0` to use all connections only one time. This parameter is of most use if the frequent changes to the IBM MQ connection parameters are expected due to redeployment of the IBM App Connect Enterprise flows, and you require the HTTP proxy servlet to reflect these changes within the specified number of seconds.

Setting up the JNDI interface for the proxy servlet
The JNDI interface for the proxy servlet requires a one time setup of the WebSphere Application Server full profile.

About this task

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

These setup tasks must all be completed in the WebSphere Application Server administrative console.

Creating a resource environment provider

About this task

Configure a resource environment provider, which encapsulates the referenceables that convert resource environment entry data into resource objects. These resource objects can then be accessed by applications.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers**.

   The **Resource environment providers** wizard opens.
2. Click **New**.

   The **Configuration** panel opens so that you can configure a new resource environment provider.
3. Type a name for the resource environment provider in the **Name** filed. For example, `MyResourceEnvironmentProvider`. It is recommended that you enter a meaningful description in the **Description** field, but it is not required. Click **OK** to continue and then save the changes.

Results

The new resource environment provider is listed in the wizard.

Creating a referenceable object

About this task

Configure a new referenceable, which specifies the factory class that converts data in the Java Naming and Directory Interface (JNDI) name space into an object that represents your resource to WebSphere Application Server.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers**.

   The **Resource environment providers** wizard opens.
2. From the **Resource environment providers** panel, select the provider that you created in the previous task: In this example, it is called *MyResourceEnvironmentProvider*.
3. Click **Referenceables**.

   The **Referenceables** panel opens.
4. Click **New**.

   The **Configuration** panel opens.
5. In the **Configuration** panel, type the following values:

   a) In the **Factory class name** field, type: `com.ibm.broker.httpproxy.MQResourceConnectionFactory`

   b) In the **Class name** field, type: `com.ibm.broker.httpproxy.MQResourceConnection`
6. Click **OK** and save the changes.

Creating resource environment entries

About this task

Configure resource environment entries, which are objects that contain information about a resource, and represent it in the JNDI name space.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers** > ***MyResourceEnvironmentProvider***.

   The **Resource environment providers** wizard opens at the configuration panel for your provider.
2. Click **Resource environment entries**.

   The **Resource environment entries** panel opens.
3. Click **New**.

   The **Configuration** panel opens.

4. In the **Configuration** panel, type your values. In this example, the values are:
   a) In the **Name** field, type for example: `MQResourceReference`
   b) In the **JNDI name** field, type for example: `proxyservlet/reference/MQResourceReference`.
      This JNDI name is used during application deployment resource mapping.
   c) In the **Referenceables** drop-down list, ensure that `com.ibm.broker.httpproxy.MQResourceConnectionFactory` is selected.
5. Click **OK** and save the changes.

Creating custom properties or define the parameters to be configured

About this task

Specify custom properties that your enterprise information system (IES) requires for the resource providers and resource factories that you configure. For example, most database vendors require extra custom properties for data sources that access the database.

Procedure

1. Select **Resources** > **Resource Environment** > **Resource Environment Providers** > ***MyResourceEnvironmentProvider*** > **ResourceEnvironment Entries** > ***MQResourceReference***. where *MyResourceEnvironmentProvider* and *MQResourceReference* are your own specified values.
   The **Resource environment providers** wizard opens at the configuration panel for your entry.
2. Click **Custom properties**.
   The **Configuration** panel opens.
3. Click **New**.
4. In the **Configuration** panel, you must type your values for all the resources that are administered through the administrative console. For example: The integration node name, the configuration file path, the client mode channel name, and the client mode port number. For each resource, you must provide values for **Name**, **Value**, **Description**, and **Type**. The following example is for the integration node name:

Option	Description
Name	This field refers to the value in the "`<param-name> </param-name>`" tag in the `web.xml`. For example: `brokerName`
Value	This field refers to the value in the "`<param-value> </param-value>`" tag in the `web.xml`. For example: RRB
Description	Optional. It is recommended that you provide a meaningful description.
Type	Specify the type. For example: `java.lang.String`

5. Click **OK** to save the configuration.
6. Click **New** to create a new configuration for the remaining resources.

Results

You have a list of configured resources that is similar to the following table, but with the values that you specified:

Name	Value	Description	Required
brokerName	RRB	Integration node name	false

Name	Value	Description	Required
configFilePath	C:\\Documents and Settings\\All Users.WINDOWS\ \Application Data\\IBM\\MQSI\\components\\RRB\ \config\soapplugin6.conf	Configuration file path	false
clientModeChannelName	SYSTEM.DEF.SVR.CONN	Client mode channel name	false
clientModePortNumber	2908	Client mode port number	false

The WebSphere Application Server wizard does not provide an option to specify the **Required** attribute and so the default value is set to false. This attribute can be ignored.

What to do next

After you complete these tasks, you must deploy the `proxyservlet.war` file. During the deployment, you must provide the JNDI reference name for the **`Resource Environment`** reference that is defined in the `web.xml` file. For example: `proxyservlet/reference/MQResourceReference`.

After the deployment, proxyservlet gives precedence to the values configures in the Resource Environment Entries. If there is an environment change, the values must be modified in the custom properties of the Resource Environment Entries. After you modify a value, restart the `Proxyservlet` application in WebSphere Application Server for the new values to take effect.

Deploying the HTTP proxy servlet on a servlet container

Load and install the HTTP proxy servlet file on a servlet container, such as Apache Tomcat, WebSphere Application Server Liberty Profile, or an application server such as WebSphere Application Server.

Before you begin

Before you deploy the HTTP proxy servlet, you must complete the following tasks:

- Enable the integration node listener for each integration server where message flows with SOAP nodes are deployed. See "HTTP listeners" on page 795 and "Switching from embedded listeners to an integration node listener" on page 798.
- "Configuring the HTTP proxy servlet" on page 149

About this task

Supported operating systems

This documentation describes the installation on Windows, but the HTTP proxy servlet can be installed on any operating system that is supported by the servlet container.

Prerequisite components

In addition to IBM App Connect Enterprise, the HTTP proxy servlet requires the following components:

- A supported version of IBM MQ. For the latest details of all supported levels of hardware and software, see the IBM App Connect Enterprise system requirements website.
- A servlet container, such as WebSphere Application Server Liberty Profile or Apache Tomcat, or an application server, such as WebSphere Application Server.

The following topics describe how to deploy the HTTP proxy servlet onto WebSphere Application Server Liberty Profile, Apache Tomcat, and WebSphere Application Server; however, the process is similar for other servlet containers or web servers:

- "Deploying the HTTP proxy servlet on WebSphere Application Server Liberty Profile" on page 160
- "Deploying the HTTP proxy servlet on Apache Tomcat" on page 161
- "Deploying the HTTP proxy servlet on WebSphere Application Server" on page 163

Results

You have deployed the HTTP proxy servlet on a servlet container or web server

What to do next

Test the HTTP proxy servlet. For information about how to complete this task, see "Testing the HTTP proxy servlet" on page 164.

Deploying the HTTP proxy servlet on WebSphere Application Server Liberty Profile
Load and install the HTTP proxy servlet file on WebSphere Application Server Liberty Profile.

Before you begin

Before you deploy the HTTP proxy servlet, you must complete the following tasks:

- "Configuring the HTTP proxy servlet" on page 149
- Enabling the integration node listener for SOAP nodes

About this task

Install and customize a WebSphere Application Server Liberty Profile (Liberty) server, and then deploy the HTTP proxy servlet by completing the following steps. The instructions assume that IBM App Connect Enterprise is installed and running on the same machine as the Liberty server.

Procedure

To deploy the HTTP proxy servlet on the Liberty server, complete the following steps:

1. Follow the instructions on the WebSphere Application Server Liberty Profile download page to download and install a Liberty server.
2. Create the following sub directory in your Liberty server installation: *Liberty_installation_path*\wlp\usr\shared\resources\wmq.
3. Copy the following IBM MQ JAR files from *WebSphere_MQ_installation_path*\Java\lib to *Liberty_installation_path*\wlp\usr\shared\resources\wmq:
   - com.ibm.mq.commonservices.jar
   - com.ibm.mq.headers.jar
   - com.ibm.mq.jar
   - com.ibm.mq.jmqi.jar
   - com.ibm.mq.pcf.jar
   - connector.jar

   The HTTP proxy servlet uses IBM MQ to communicate with IBM App Connect Enterprise.
4. Copy the HTTP proxy servlet .war file that you exported from IBM App Connect Enterprise (for example, myproxyservlet.war) to *Liberty_installation_path*\wlp\usr\shared\apps.
5. In the Eclipse workspace for the Liberty server, click the **Servers** tab and open the server.xml file by double-clicking the **Server Configuration** entry.
6. Add the following lines to server.xml to define the shared library that contains the IBM MQ .jar files:

```
<library id="wmq" name="wmq">
     <fileset dir="${shared.resource.dir}/wmq" includes="*.jar"/>
</library>
```

7. Add the following lines to `server.xml` to set the context root for the HTTP proxy servlet and link the HTTP proxy servlet with the IBM MQ shared library. For information about the context root, see "Importance of the context root when you configure the HTTP proxy servlet" on page 147.

```
<application id="proxyservlet" location="servlet_file_name.war"
        name="proxyservlet" type="war" context-root="context_root">
        <classloader privateLibraryRef="wmq"/>
</application>
```

   where:

   servlet_file_name
   Specifies the name of the HTTP proxy servlet file that you exported from IBM App Connect Enterprise.

   context_root
   Specifies the context root you want to assign to the HTTP proxy servlet.

   Note: If you want the HTTP proxy servlet to use an empty context root set `context-root="/"`.

8. Save and close `server.xml`, and start the Liberty server.
9. To test if the HTTP proxy servlet is active, enter the following URL in a web browser:

```
http://host_name:port/context_root
```

   where:

   host_name
   Specifies the host name of your Liberty server.

   port
   Specifies the port number of your Liberty server.

   context_root
   Specifies the context root that you assigned to the HTTP proxy servlet.

   For example: `http://localhost:9080/`.

   If the HTTP proxy servlet can reach the integration node, then you should get an HTTP Status 404 response with the following error message:

   `URI / does not map to any message flow in integration node integration_node`

   where *integration_node* is the name of your integration node.

Results

The HTTP proxy servlet is deployed on the WebSphere Application Server Liberty Profile server.

What to do next

The HTTP proxy servlet is now ready to be tested with an integration node that receives HTTP (not HTTPS) requests and passes them to a message flow. For information about how to complete this task, see "Testing the HTTP proxy servlet" on page 164.

Deploying the HTTP proxy servlet on Apache Tomcat
Load and install the HTTP proxy servlet file on Apache Tomcat.

Before you begin

Before you deploy the HTTP proxy servlet, you must complete the following tasks:

- "Configuring the HTTP proxy servlet" on page 149
- Enabling the integration node listener for SOAP nodes

About this task

Install and customize Apache Tomcat, and then deploy the HTTP proxy servlet by completing the following steps. The instructions assume that IBM App Connect Enterprise is installed and running on the same machine as Apache Tomcat.

Procedure

1. Download Apache Tomcat from the Apache Tomcat 9 download page.
2. Run the Apache Tomcat installer and follow the instructions to complete the installation.

   Note: When prompted for the path to a Java virtual machine (JVM) you can use the JVM that is deployed with IBM App Connect Enterprise, for example `C:\Program Files\IBM\ACE\12.0.n.0\common\jdk\jre`.
3. Find the file `catalina.properties`.

   This can be found at the following location:

   Tomcat_installation_path`/conf/catalina.properties`

   where *Tomcat_installation_path* is the name of your Apache Tomcat installation directory.
4. Edit `catalina.properties` and locate the following line:

   ```
   shared.loader=
   ```
5. Edit the line so that it includes the following text:

   ```
   shared.loader=${catalina.home}/shared/lib,${catalina.home}/shared/lib/*.jar
   ```
6. Create the directory *Tomcat_installation_path*`/shared/lib`.
7. Copy the following IBM MQ JAR files from *WebSphere_MQ_installation_path*`\Java\lib` to *Tomcat_installation_path*`\shared\lib`:

   - `com.ibm.mq.commonservices.jar`
   - `com.ibm.mq.headers.jar`
   - `com.ibm.mq.jar`
   - `com.ibm.mq.jmqi.jar`
   - `com.ibm.mq.pcf.jar`
   - `connector.jar`

   The HTTP proxy servlet uses IBM MQ to communicate with IBM App Connect Enterprise.
8. Start the Apache Tomcat Windows service.
9. Open a web browser and enter the web address `http://localhost:`*port* where *port* is the HTTP port number you specified in the Apache Tomcat installation.

   The default value for the port is 8080.

   The Apache Tomcat home page is displayed.
10. Click **Tomcat Manager** and enter the admin user ID and password.
11. Scroll down to the section **WAR file to deploy** and click **Browse**.
12. Navigate to the HTTP proxy servlet file that you exported from IBM App Connect Enterprise. For example, `myproxyservlet.war`.

    Note: By default in Apache Tomcat, the context root that is used by the HTTP proxy servlet is the same as the `.war` file name. You can rename the `.war` file before you deploy the HTTP proxy servlet in order to assign a different context root to the HTTP proxy servlet. For information about the context root, see "Importance of the context root when you configure the HTTP proxy servlet" on page 147. If you want the HTTP proxy servlet to use an empty context root (`/`), then you must rename the `.war` file to `ROOT.war` and uninstall the current default web application (the Tomcat Welcome page).
13. Click **Deploy**.

14. To test if the HTTP proxy servlet is active, enter the following URL in a web browser:

    ```
    http://host_name:port/context_root
    ```

    where:

    host_name
    Specifies the host name of your Apache Tomcat server.

    port
    Specifies the port number of your Apache Tomcat server.

    context_root
    Specifies the context root that you assigned to the HTTP proxy servlet.

    For example: `http://localhost:8080/`.

    If the HTTP proxy servlet can reach the integration node, you should get an `HTTP Status 404` response with the following error message:

    `URI / does not map to any message flow in integration node integration_node`

    where *integration_node* is the name of your integration node.

Results

The proxy servlet is deployed on your Apache Tomcat server.

What to do next

The HTTP proxy servlet is now ready to be tested with an integration node that receives HTTP (not HTTPS) requests and passes them to a message flow. For information about how to complete this task, see “Testing the HTTP proxy servlet” on page 164.

Deploying the HTTP proxy servlet on WebSphere Application Server
Load and install the HTTP proxy servlet file on WebSphere Application Server.

Before you begin

Before you deploy the HTTP proxy servlet, you must complete the following tasks:

- Configure the HTTP proxy servlet, as described in “Configuring the HTTP proxy servlet” on page 149
- Enable the integration node listener, as described in Enabling the integration node listener for SOAP nodes
- Identify or install a WebSphere Application Server installation that can host the HTTP proxy servlet.

Procedure

To deploy the HTTP proxy servlet on WebSphere Application Server, complete the following steps:

1. Deploy the HTTP proxy servlet file that you exported from IBM App Connect Enterprise, by following the instructions for deploying a web application in the WebSphere Application Server Version 8.5 product documentation online.

   Ensure that you select the correct context root for your HTTP proxy servlet, as described in “Importance of the context root when you configure the HTTP proxy servlet” on page 147.

2. To test if the HTTP proxy servlet is active, enter the following URL in a web browser:

   ```
   http://host_name:port/context_root
   ```

   where:

   host_name
   Specifies the host name of your WebSphere Application Server

   port
   Specifies the port number of your WebSphere Application Server

   context_root
   Specifies the context root that you assigned to the HTTP proxy servlet

   For example: `http://localhost:9080/`.

   If the HTTP proxy servlet can reach the integration node, you will see an `HTTP Status 404` response with the following error message:

   `URI / does not map to any message flow in integration node` *integration_node*

   where *integration_node* is the name of your integration node.

Results

The HTTP proxy servlet is deployed on WebSphere Application Server.

What to do next

The HTTP proxy servlet is now ready to be tested with an integration node that receives HTTP (not HTTPS) requests and passes them to a message flow. For information about how to complete this task, see "Testing the HTTP proxy servlet" on page 164.

Testing the HTTP proxy servlet

Test the HTTP proxy servlet with an integration node that receives HTTP requests and passes them to a message flow.

Before you begin

To test the HTTP proxy servlet, use an existing web services client application or write your own SSL test client application by using Java.

Before you test the HTTP proxy servlet, you must have completed the following tasks:

- "Configuring the HTTP proxy servlet" on page 149
- Enabling the integration node listener for SOAP nodes for the proxy servlet to access
- "Deploying the HTTP proxy servlet on a servlet container" on page 159

Procedure

To test the HTTP proxy servlet, complete the following steps:

1. Install a client application that can send HTTP or SSL (HTTPS) requests.

   For example,:

   - An open source tool available from the OpenSSL downloads website that can be used to send HTTPS requests to the servlet container.
   - An integration node message flow that has an HTTPRequest or SOAPRequest node, which can generate and send HTTP requests to an HTTP listener.
   - A web browser, by using web pages or Java Server Pages (JSP), that can send HTTP POST requests. Most web browsers support HTTP and HTTPS.
   - A client application that sends requests by using HTTP, HTTPS, or both HHTP and HTTPS.

2. Configure a message flow with HTTP and SOAP input and reply nodes. The HTTPInput and SOAPInput nodes must be configured with a URL that matches the context root of the HTTP proxy servlet; for more information, see “Importance of the context root when you configure the HTTP proxy servlet” on page 147.

   The message flow receives the messages from the HTTP proxy servlet. If an HTTP or SOAP reply node is configured, responses are sent back to the HTTP proxy servlet.

Results

You have tested the HTTP proxy servlet.

Deleting an integration node

Delete an integration node by either using the command line on the system where IBM App Connect Enterprise is installed, or use the IBM App Connect Enterprise Toolkit.

Before you begin

Check that your user ID has the correct authorizations to delete the integration node; for details, see Security requirements for administrative tasks.

On Windows, Linux, and UNIX systems, you must set up your command-line environment before you delete an integration node, by running the product profile or console; see “Setting up a command environment” on page 64.

About this task

On Windows, Linux, and UNIX systems, you can delete an integration node by using the IBM App Connect Enterprise Toolkit.

You cannot delete a remote integration node by using the IBM App Connect Enterprise Toolkit, but you can remove the connection to the remote integration node from your workspace. To remove the connection to a remote integration node in the IBM App Connect Enterprise Toolkit, right-click the integration node in the Integration Explorer view, and click **Remove Connection**.

You can delete an integration node in the following ways:

- “Deleting an integration node by using the command line” on page 165
- “Deleting an integration node by using the IBM App Connect Enterprise Toolkit” on page 166.

Deleting an integration node by using the command line

Procedure

1. Stop the integration node by using the **mqsistop** command.
2. Delete the integration node by doing one of the following, dependent on your operating system:
   - Linux Windows UNIX Enter the following command to delete the integration node:

     ```
     mqsideletebroker INODE
     ```

     where INODE is the integration node name.
3. If you created integration server profiles for this integration node, delete these profiles if they are no longer required.
4. Any empty integration server `shared-classes` directories (under *workpath*/`config/`*<my_int_node_name>*`/`*<my_int_server_label>*) are automatically deleted. Otherwise, if no longer required, it is left to the user to manually delete the directories and contents.

5. If the integration node level `shared-classes` directory (under *workpath*`/config/`*<my_int_node_name>*) is empty, it is automatically deleted. Otherwise, if no longer required, it is left to the user to manually delete the directory and contents.

Results

On completion of this task:

- Linux UNIX On Linux, and UNIX, the integration node definition is removed.
- Windows Stopped and deleted the Windows service that runs the integration node.
- Removed any integration server profile scripts that are no longer needed.
- Removed any empty integration server `shared-classes` directories.
- Removed any empty integration node `shared-classes` directory.

Deleting an integration node by using the IBM App Connect Enterprise Toolkit

Procedure

1. Open the IBM App Connect Enterprise Toolkit, and switch to the Integration Development perspective.
2. In the Integration Explorer view, right-click the integration node, and click **Delete**.
   If the integration node is a remote integration node, the connection to the integration node is removed from your workspace.
   If the integration node is a local integration node, the **Delete Local Broker** wizard is displayed.
3. Click **Finish** to delete the integration node.
   If the integration node is running, the wizard stops the integration node.

Results

You have deleted the integration node.

Configuring an integration node as an IBM MQ service

Use these topics to make changes when your integration node is operating as an IBM MQ service.

About this task

The topics in this section describe how to configure your system for high availability.

- "Starting and stopping an integration node as an IBM MQ service" on page 166

Starting and stopping an integration node as an IBM MQ service

Configuring an integration node to start and stop as an IBM MQ service.

Before you begin

Ensure that you make the mqm user ID a member of the mqbrkrs group. On Windows systems, you must restart your workstation for the change to take effect.

Procedure

To configure an integration node to run as an IBM MQ service, complete the following steps.

1. Create an integration node to start as an IBM MQ service by using the **mqsicreatebroker** command with the **-d** parameter.

   Note: You must be a member of the mqm group to run the **mqsicreatebroker** command with the **-d** parameter.

   For example,

   ```
   mqsicreatebroker MyBroker -q MyQMGR -d defined
   ```

   where

   MyBroker
   Is the name of the integration node that you want to start and stop as an IBM MQ service.

   MyQMGR
   Is the name of the queue manager that is associated with the integration node.
2. Optional: If you want to create an App Connect Enterprise vault to hold the credentials that are used by the integration node when secured resources are accessed, use the **mqsivault** command.

   For example,

   ```
   mqsivault MyBroker --create --vault-key 12345678
   ```

   where

   MyBroker
   Is the name of the integration node that you want to start and stop as an IBM MQ service.
3. Optional: If you created an IBM App Connect Enterprise vault by running the **mqsivault** command at step “2” on page 167, you must configure your environment by running one of the following steps:
   a) Set the vault key as an environment variable. For example, on Windows:`set MQSI_VAULT_KEY=12345678`
   b) Add the vault key to an `.mqsivaultrc` file and copy the file into the home directory of the user who starts the IBM MQ queue manager.

      For more information about IBM App Connect Enterprise vaults and the `.mqsivaultrc` file, see command and command.
4. Stop the queue manager.

Results

When you configure an integration node to start and stop as an IBM MQ service:

- The integration node starts and stops automatically when its associated queue manager starts and stops. For a multi-instance integration node, this action can occur during failover of the active queue manager.
- A multi-instance integration node cannot be started in standby mode when its IBM MQ service is defined as active.
- You can stop the integration node manually by using the **mqsistop** command, but the integration node does not restart until the queue manager is stopped and started again.
- On UNIX and Linux systems, the integration node environment is inherited from IBM MQ. Set any required environment variables (such as ODBCINI) by using a script in the `work_path/common/profiles` directory. For more information, see “Setting up a command environment” on page 64.

Configuring integration servers

Create and configure the integration servers that you want to use on the operating system of your choice.

Before you begin

Ensure that the following requirements are met:

- Your user ID has the correct authorizations to perform the task. The authorizations are defined in Security requirements for administrative tasks.
- You have initialized the command environment on distributed systems; see "Setting up a command environment" on page 64.

About this task

You can create an integration server by using the IBM App Connect Enterprise Toolkit, the web user interface, or App Connect Enterprise commands, and you can then configure it by modifying the `server.conf.yaml` file.

When you have created and configured your integration servers, you can administer them and their resources by using the web user interface, the command line, or programmatically by using the App Connect Enterprise REST API. For more information, see "Managing integration servers" on page 249, "Managing resources by using the administration REST API" on page 334, and "Accessing the web user interface" on page 89.

Procedure

1. Create an integration server by following in the instructions in "Creating an integration server" on page 168.
2. Modify the integration server properties by following the instructions in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

   If you use any App Connect Enterprise commands that modify the integration server, an overrides directory is created under the working directory for the integration server. This overrides directory contains an additional `server.conf.yaml` configuration file, which includes property values that have been set by commands; for example, `<work_directory>/overrides/server.conf.yaml`. The values of properties in this `overrides/server.conf.yaml` file override any values that you have set in the integration server's `server.conf.yaml` file (`<work directory>/server.conf.yaml`).
3. Optional: Configure a default application for an integration server, by following the instructions in "Configuring a default application for an integration server" on page 181.
4. Optional: If you want to delete an integration server, follow the instructions in "Deleting an integration server" on page 175.

Creating an integration server

You can create integration servers by using the IBM App Connect Enterprise Toolkit, the web user interface, or commands.

About this task

You must create an integration server before you can deploy integration solutions and related resources.

You can create one or more independent integration servers, each with their own identity, and deploy them either to containers in the cloud or in an on-premises environment. If you are planning to run App Connect Enterprise directly on a physical machine or virtual machine image, you are advised to define integration servers under an integration node, which will manage its associated integration servers.

For information about why you might want to create multiple integration servers, see Integration servers and integration nodes.

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers that you can use; see "Restrictions that apply in each operation mode" on page 5.

For an independent integration server, you create the server's work directory and then configure the server's properties, as described in "Creating an integration server work directory by using the mqsicreateworkdir command " on page 169.

You can also create and start a local independent integration server by using the IBM App Connect Enterprise Toolkit, as described in "Creating and starting a local, independent integration server by using the Toolkit" on page 170.

To create an integration server under an integration node, first start the integration node, and then use one of the following methods:

- "Creating an integration server that is managed by an integration node, by using the Toolkit" on page 171
- "Creating an integration server that is managed by an integration node, by using the web user interface" on page 171
- "Creating an integration server that is managed by an integration node, by using the mqsicreateexecutiongroup command" on page 172

You can also use the IBM Integration API to create integration servers on all platforms; see "Managing resources by using the IBM Integration API" on page 444.

Creating an integration server work directory by using the `mqsicreateworkdir` command

To create a work directory to be used by an independent integration server, use the **`mqsicreateworkdir`** command.

Procedure

1. Open a command window for your current installation.
2. Create a work directory for your independent integration server, by running the **`mqsicreateworkdir`** command, specifying the full path to the directory that you want to create.
   For example:

   ```
   mqsicreateworkdir c:\myaceworkdir\myserver
   ```

   This command creates the specified work directory, which contains a default configuration file called `server.conf.yaml`. This YAML file contains the default settings for your new integration server. The command also creates subdirectories, which will be used by the integration server when it is running. These include a `log` directory and a `run` directory. The `log` directory contains files of log messages, which can be used to review the status of your integration server. The `run` directory is where you can place your BAR files prior to starting your integration server, or while it is running. The resources from the BAR file are extracted and started by the integration server.

   If you have a BAR file with resources that contain graphical data maps, XML schemas, or DFDL schemas, these resources can be compiled using the command, prior to starting the integration server. This precompilation reduces the time that it takes for the deployed application to start processing messages.

   See the **`mqsicreateworkdir`** command for more information.
3. Configure your integration server by setting properties in the `server.conf.yaml` file, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Results

The integration server is created and the updated `server.conf.yaml` configuration file is stored in integration server's subdirectory in the file system.

What to do next

You can now start the integration server by running the command, as described in "Starting an integration server" on page 250.

You can deploy resources to your integration server, as described in Chapter 7, “Deploying integration solutions,” on page 2463.

Creating and starting a local, independent integration server by using the Toolkit

You can use the IBM App Connect Enterprise Toolkit to create and start a local, independent integration server.

About this task

You can create and start a local, independent integration server by using a wizard in the IBM App Connect Enterprise Toolkit. The wizard starts a process on the operating system for the independent integration server. The Toolkit does not manage the process for an independent integration server in the same way that an integration node manages the integration servers that are associated with it. When you stop the Toolkit, any independent integration server processes that were started from the Toolkit are stopped. However, if the Toolkit stops unexpectedly, or if its Eclipse process is stopped, the independent integration server processes continue to run on your operating system, and you will need to use an operating system process management tool to stop them.

Procedure

Create and start a local, independent integration server by completing the following steps:

1. In the Integration Explorer view, right-click **Integration servers** and then click **Create a local integration server**.
2. In the **Connection details** wizard, enter the following information:

   - **Name**

     Enter the name of the local independent integration server that you want to create and start. When you run the wizard for the first time, the default name for the integration server is TEST_SERVER. If you run the wizard again, the TEST_SERVER will already exist so you must specify an alternative name for the new integration server.

   - **REST Administration Port**

     Specify the port to be used for REST administration. By default, you can allow the wizard to find a port that is available to use. If you want to specify a different port, first ensure that it is not being used by another process, then deselect the checkbox and enter the required port number. The port that you specify cannot be used by the integration server if it is already in use by another process.

   - **HTTP Port**

     Specify the port that is used by the HTTPConnector resource manager when an HTTP-based message flow is deployed as part of an application. By default, you can allow the wizard to find a port that is available to use. If you want to specify a different port, first ensure that it is not being used by another process, then deselect the checkbox and enter the required port number. The port is then used when the message flow is deployed. The wizard finds an available port when the integration server is started. If the message flow is deployed at a much later date, the port might no longer be available, in which case an error message is shown at deployment time to indicate that the HTTP listener could not be started for that port. If this happens, specify the correct port used by the HTTPConnector in the `server.conf.yaml` file in the `overrides` directory, and then stop and start the integration server.

   - **JVM Debug Port**

     Specify the port to be used for debugging a message flow. By default, you can allow the wizard find a port that is available to use. If you want to specify a different port, first ensure that it is not being used by another process, then deselect the checkbox and enter the required port number. This sets the **`jvmDebugPort`** property in the `server.conf.yaml` file for the JVM resource manager to use.

If you do not want to debug a message flow, you can disable the JVM debug port by deselecting the checkbox and entering the value `0` in the field.

- Click **Finish**.

  A confirmation message shows that a configuration directory has been created in the workspace and that the integration server has been started using the details in the configuration directory. The configuration directory that is created in your workspace has the same name as the integration server that has been created and started. This is the same configuration directory that is created by the **`mqsicreateworkdir`** command. The configuration directory is shown under **Independent Resources** in your workspace. The wizard creates an override file for the `server.conf.yaml` that is used by the integration server. This override file contains the values of the ports that will be used by the integration server.

3. Optional: When you have created and started the integration server, you can deploy an application or library to it.
4. Optional: You can view the `console.log` file for the integration server. The location of the `console.log` file is under the main configuration directory.
5. Optional: You can stop the integration server by right-clicking it and then clicking **Stop**, which stops the integration server process. You can restart the integration server by right-clicking it and then clicking **Start** to start the integration server process.
6. Optional: You can remove the connection for the integration server when it has been stopped, but the configuration directory is not deleted from your workspace automatically; however, you can delete the directory manually.

   If you have previously stopped the integration server and removed the connection for it, you can create it again and it will use the configuration directory in your workspace, if it exists. In this case, a message informs you that the existing configuration directory in your workspace will be used.

Creating an integration server that is managed by an integration node, by using the Toolkit

Procedure

To create an integration server under an integration node, you can use the IBM App Connect Enterprise Toolkit:

1. Optional: If you are working with a remote integration node, connect to the integration node; see Connect to a remote integration node.
2. Make sure that the integration node is running.
3. In the Integration Explorer view, right-click the integration node, and then select **New Integration Server**.
4. In the **New Integration Server** dialog, enter the integration server name.
5. Click **OK** to create the integration server.

Results

The integration server is added to the selected integration node.

Creating an integration server that is managed by an integration node, by using the web user interface

Procedure

To create an integration server under an integration node, you can use the web user interface:

1. Make sure that the integration node is running.
2. Start the web user interface, as described in “Accessing the web user interface” on page 89.

3. Select the **Servers** tab, and then click **Create**.
4. In the **New Integration Server** dialog, enter the integration server name.
5. Click **OK** to add the integration server.
6. Optional: Configure your integration server by setting properties in the `server.conf.yaml` file, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Results

The integration server is created.

Creating an integration server that is managed by an integration node, by using the `mqsicreateexecutiongroup` command

To create integration server under an integration node, you can use the **mqsicreateexecutiongroup** command.

Procedure

1. If you are creating an integration server on Linux or Windows systems:
   a) Open a command window for your current installation.
   b) Enter the **mqsicreateexecutiongroup** command, and specify the parameters for the integration server that you want to create.
      - If the integration node is local, specify the integration node name. For example:

        ```
        mqsicreateexecutiongroup INODE -e IServer_2
        ```

      - If the integration node is remote, you can specify a configuration file. For example:

        ```
        mqsicreateexecutiongroup -n INODE.broker -e IServer_2
        ```

      - If the integration node is remote, you can alternatively specify the connection parameters **-i** and **-p**. For example:

        ```
        mqsicreateexecutiongroup -n INODE -e IServer_2 -p 4414 -i host
        ```

      See the **mqsicreateexecutiongroup** command description for more details about these options.
2. If you are creating an integration server on z/OS:
   a) Configure the BIPCREG job to specify the properties for the integration server that you want to create.
   b) Run the BIPCREG job.
3. Optional: Configure your integration server by setting properties in the `server.conf.yaml` file, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Results

The integration server has been created under the specified integration node.

What to do next

Deploy resources to your integration server; see Chapter 7, "Deploying integration solutions," on page 2463.

Configuring an integration server by modifying the server.conf.yaml file

You can configure your IBM App Connect Enterprise integration server by modifying properties in a `server.conf.yaml` configuration file. The location of the `server.conf.yaml` file that you need to

modify depends on whether you are configuring an independent integration server or an integration server that is managed by an integration node.

Before you begin

Ensure that you set up your command environment, as described in "Setting up a command environment" on page 64.

About this task

When you have created an integration server, you set properties in the `server.conf.yaml` file to configure the operation of the integration server and associated resources. For example, you can set a REST administration port and an HTTPS port, and you can configure the trace level, activity logging, JVM, and the reporting of statistics data for your integration server. You can also configure the integration server to record all messages that pass through a message flow, and then use these recorded messages to generate unit tests.

For an independent integration server, the `server.conf.yaml` configuration file is created for you automatically when you use the **`mqsicreateworkdir`** command to create an integration server work directory. The `server.conf.yaml` file is created in the root of the specified work directory: `<work directory>/server.conf.yaml`.

If you use any commands that modify the integration server, an overrides directory is created under the working directory for the integration server. This overrides directory contains an additional `server.conf.yaml` configuration file, which contains property values that are set by commands; for example, `<work directory>/overrides/server.conf.yaml`. The values of properties in this `overrides/server.conf.yaml` file override any values that you have set in the integration server's `server.conf.yaml` file (`<work directory>/server.conf.yaml`).

If a property has been set in the integration server's `server.conf.yaml` file, and also in the overrides directory (`/overrides/server.conf.yaml`), the property value that has been set in the overrides directory is used. Therefore, if an integration server does not appear to be using the settings that you would expect, check the `server.conf.yaml` file in the overrides directory to see if your expected property value has been overridden by a command. If you want to manually override the settings that have resulted from a command, you can either edit the property in the `server.conf.yaml` file in the overrides directory, or you can remove the entry from the overrides directory and modify the base `server.conf.yaml` file instead.

For integration servers that are managed by an integration node, each server has its own `server.conf.yaml` configuration file that overrides common settings from the integration node's `node.conf.yaml` configuration file. When you create an integration node, the `node.conf.yaml` file is located at: `$MQSI_WORKPATH/components/<Node name>/node.conf.yaml`.

If you use commands to modify the integration node, the changes are saved in the integration node's override `node.conf.yaml` file. This file is located at: `$MQSI_WORKPATH/components/<Node name>/overrides/node.conf.yaml`, as described in "Creating an integration node by using the command line" on page 115. When you create a managed integration server for an integration node, server-specific settings are created for it in its own `server.conf.yaml` file. The server-specific file is located at: `$MQSI_WORKPATH/components/<Node name>/servers/<Server name>/server.conf.yaml`.

If you use commands to modify this integration server, the changes are saved in: `$MQSI_WORKPATH/components/<Node name>/servers/<Server name>/overrides/server.conf.yaml`. The values of properties in this `overrides/server.conf.yaml` file override any values that you have set in the `<Node name>/servers/<Server name>/server.conf.yaml` file.

Procedure

1. If the integration server does not exist already, you can create it by following the instructions in "Creating an integration server" on page 168.

2. Use a YAML editor to open the `server.conf.yaml` file.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your integration server configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
3. Modify the properties that you want to change in the file.
   For example:

   - In the RestAdminListener section, set a value for the **Port** property to be used by the REST administration port, which is the primary method of communicating with the integration server. (Default value of 7600.)
   - In the ResourceManagers / HTTPConnector section, set a value for the **ListenerPort** so that you can send messages to a flow that is using a HTTPInput node. (Default value of 7800.)
   - In the ResourceManagers / JVM section, set a value for the **jvmDebugPort** so that you can use the Flow Debugger. For example, set this property to 6511.

   You can also configure the integration server to record all messages that pass through a message flow, by setting properties in the **RecordedMessageManager** section of the `server.conf.yaml` file. You can then use these recorded messages to generate unit tests for the flow, as described in "Generating tests from recorded messages in a message flow" on page 1908.
4. Save your changes to the `server.conf.yaml` file.

   If you run any commands that modify the integration server at any time after you set properties in the `server.conf.yaml` file, an overrides directory is created under the working directory for the integration server. This overrides directory contains another `server.conf.yaml` configuration file, which contains property values that are set by commands; for example, `<work directory>/overrides/server.conf.yaml`. The values of properties in this `overrides/server.conf.yaml` file override any values that you have set in the integration server's `server.conf.yaml` file (`<work directory>/server.conf.yaml`). If you want to manually override the settings that have resulted from a command, you can either edit the property in the `server.conf.yaml` file in the overrides directory, or you can remove the entry from the overrides directory and modify the base `server.conf.yaml` file instead.
5. Specify the security credentials to be used by the integration server when you connect to a secured resource (such as a database) by using the **mqsisetdbparms** command. Use the **-w** parameter to specify the integration server's work directory that you created in step 1:
   For example:

   ```
   mqsisetdbparms -w c:\myaceworkdir -n jdbc::secID -u iibuser -p password
   ```

   When you run this command, the user ID and password are stored securely in the IBM App Connect Enterprise credentials store. For more information about using the **mqsisetdbparms** command, see command.

   Alternatively, you can configure an independent integration server to connect to secured resources by using credentials that are stored in encrypted form in an App Connect Enterprise vault. You can configure security credentials by using the command or the administrative REST API. These encrypted credentials are stored in the integration server's vault, which you configure by using the command. For more information, see "Configuring encrypted security credentials" on page 177.
6. Optional: You can configure applications to run when the integration server starts, by pre-loading the BAR file into the `run` folder of the integration server's work directory. For information, see command.
7. Restart the integration server. The properties that you set in the `server.conf.yaml` file take effect when the integration server is started. If you modify these properties again, you must start the integration server again for the subsequent changes to take effect. For more information, see "Starting an integration server" on page 250.

   If the integration server fails to restart, check to see whether the failure was caused by a YAML parsing error. For an independent integration server, parsing errors are reported to the `stderr` log. For an

integration server that is managed by an integration node, YAML parsing errors are reported to the integration server's `console.txt` file. For more information, see Standard system logs.

Deleting an integration server

Delete integration servers by using IBM App Connect Enterprise Toolkit, the web user interface, or a command.

Before you begin

Check that the integration server is running; you cannot delete an integration server that is stopped.

About this task

Use one of the following methods to complete this task:

- "Deleting an integration server by using the IBM App Connect Enterprise Toolkit" on page 175
- "Deleting an integration server by using the web user interface" on page 175
- "Deleting an integration server by using the mqsideleteexecutiongroup command" on page 176

You can also use the IBM Integration API to delete integration servers on all platforms; see "Managing resources by using the IBM Integration API" on page 444.

Deleting an integration server by using the IBM App Connect Enterprise Toolkit

Procedure

Complete the following steps to delete an integration server:

1. Open the IBM App Connect Enterprise Toolkit, and switch to the Integration Development perspective.
2. In the Integration Explorer view, expand your integration node.
3. Right-click the integration server that you want to delete, and click **Delete** > **Integration Server**.

Results

Your integration server and all resources that are deployed to it are deleted.

- If you created any integration server profiles for this integration server, delete them manually if they are no longer required.
- If the `shared-classes` directory (under *workpath*`/config/`*<my_int_node_name>*`/`*<my_int_server_label>*) is empty, this directory is automatically deleted.

  Note: If the `shared-classes` directory is not empty but is no longer required, the directory can be deleted manually.

Deleting an integration server by using the web user interface

Procedure

Complete the following steps to delete an integration server:

1. Start the web user interface for your integration node; see "Accessing the web user interface" on page 89.

   The navigator is displayed on the left side of the pane, showing the servers (integration servers), message flows, and other resources that are owned by your integration node.
2. Expand the **Servers** folder, click the down arrow beside the integration server you want to delete to display the menu, and then click **Delete**.

3. Click **Yes** to confirm that you want to delete the integration server.

Results

Your integration server and all resources that are deployed to it are deleted.

- If you created any integration server profiles for this integration server, delete them manually if they are no longer required.
- If the `shared-classes` directory (under *workpath*`/config/`*<my_int_node_name>*`/`*<my_int_server_label>*) is empty, this directory is automatically deleted.

  Note: If the `shared-classes` directory is not empty but is no longer required, the directory can be deleted manually.

Deleting an integration server by using the `mqsideleteexecutiongroup` command

Procedure

Complete the following steps to delete an integration server:

1. If you are deleting an integration server on Linux, UNIX, and Windows systems:
   a) Open a command window for your installation.
   b) Enter the **mqsideleteexecutiongroup** command, and specify the parameters for the integration server that you want to delete.
      - If the integration node is local, specify the integration node name; for example:

        ```
        mqsideleteexecutiongroup -n INODE -e IServer_2
        ```

      - If the integration node is remote, you can specify a configuration file; for example:

        ```
        mqsideleteexecutiongroup -n INODE.broker -e EGroup_2
        ```

      - If the integration node is remote, you can alternatively specify the connection parameters **-i** and **-p**; for example:

        ```
        mqsideleteexecutiongroup -e IServer_2 -n INODE -e IServer_2 -p 4414 -i
        ```

      See the **mqsideleteexecutiongroup** command description for more details about these options.
2. If you are deleting an integration server on z/OS:
   a) Configure the BIPDLEG job to specify the properties for the integration server that you want to delete.
   b) Run the BIPDLEG job.

Results

Your integration server and all resources that are deployed to it are deleted.

- If you created any integration server profiles for this integration server, delete them manually if they are no longer required.
- If the `shared-classes` directory (under *workpath*`/config/`*<my_int_node_name>*`/`*<my_int_server_label>*) is empty, this directory is automatically deleted.

  Note: If the `shared-classes` directory is not empty but is no longer required, the directory can be deleted manually.

Configuring encrypted security credentials

You can configure integration nodes and integration servers to connect to secured resources by using credentials that are stored in encrypted form in an IBM App Connect Enterprise vault.

You can configure security credentials by using the command or the administrative REST API, and you can view credentials by using the web user interface. The encrypted credentials are stored in a vault, which you can configure by using the command, or by specifying a vault key on the command.

Alternatively, you can use the **mqsisetdbparms** command to associate credentials with resources that are accessed by an integration server or an integration node. For more information, see command.

Video: Storing encrypted security credentials in a vault using App Connect Enterprise

This video demonstrates create a vault in App Connect Enterprise and then store credentials in it in encrypted form.

Note:

This video was created for App Connect Enterprise 11.0, but also applies to App Connect Enterprise 12.0.

Creating a vault by using the `mqsivault` command

Before you can store encrypted credentials for an integration node or integration server, you must configure an App Connect Enterprise vault. You create a separate vault for each independent integration server, and for each integration node. Each independent integration server has its own vault, with its own vault key. Each integration node has its own vault, with its own vault key, which is shared by all the integration servers that it manages. Each integration server that is managed by an integration node has its own credentials stored in the vault, but all the credentials in the vault are accessed by the same vault key.

You can use the **mqsivault** command to create or destroy a vault, to change or verify a vault key, or to retrieve credentials from the vault. The vault stores the credentials (in encrypted form), and the integration node or server uses them to access secured resources.

For more information about how to use the command, see command.

Creating a vault by using the `mqsicreatebroker` command

If you create an integration node by running the **mqsicreatebroker** command, you can create a vault for that integration node by specifying either the **--vault-key** or **--vaultrc-location** parameter on the command. For more information about how to use the command, see command.

Configuring encrypted credentials by using the `mqsicredentials` command

You can use the **mqsicredentials** command to configure an integration node or integration server to use encrypted credentials for connecting to a secured resource.

You use the **mqsicredentials** command to create, update, report, and delete credentials for an independent integration server or for an integration node and the integration servers that it manages. You can use the command to create and report credentials when the integration server is running or stopped, but you must stop the integration server before you can update or delete credentials.

For information about how to use the command, see command.

Creating and viewing credentials by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to create or report security credentials for an integration node or server. For information about using the administration REST API, see REST API for administering integration servers.

Viewing credentials by using the web user interface

You can use the IBM App Connect Enterprise web user interface to view credentials for an integration node or server.

To display information about the credential, start the web user interface to view the relevant integration server, and then click the tile for the credential that you want to view. The properties for that credential are displayed, including the user name, authentication type, credentials provider, whether the credential is read-only, and whether a password has been set. For information about how to start the web user interface, see “Accessing the web user interface” on page 89.

Configuring security credentials for an independent integration server in the server.conf.yaml file

You can configure an independent integration server to connect to secured resources by using credentials that are defined in the integration server's `server.conf.yaml` configuration file.

You can configure credentials for each credential type by setting properties in the **`ServerCredentials`** section of the `server.conf.yaml` file. You can specify the credential type, name, and each of the properties that you want to set; for example:

```
ServerCredentials:
  odbc:
    USERDB:
      username: "user1"
      password: "pass"
    DATAFLOW:
      username: "usr2"
      password: "myLongPassword"
  salesforce:
    mysf1:
      username: "aName"
      password: "passw0rd"
      clientId: "clientEye"
      clientSecret: "worldsBiggestSecret"
  jdbc:
    db:
      username: "myusername"
```

If you want to change the credentials when they have been set, you must update the properties in the `server.conf.yaml` file and then restart the integration server. The credentials are read when the integration server starts, and can be used by the message flows that are running on the integration server.

The credentials are accessible through the web user interface, the REST API, and the **`mqsicredentials`** command (when the integration server is running). The **`credentialProvider`** field for credentials that are defined in this way is shown as `servercredentials`.

You can specify the following credential types and properties in the `server.conf.yaml` file:

- **cd**:

  Specify this type to set credentials for connecting an IBM Sterling Connect:Direct® CDOutput node to its Connect:Direct server.

  You can set the **`username`** and **`password`** properties for connecting to a Connect:Direct server.

- **cics**:

  Specify this type to set credentials for connecting a CICSRequest node to a CICS® Transaction Server for z/OS server.

  You can set the **`username`** and **`password`** properties for connecting to a CICS Transaction Server for z/OS server. Password is optional.

- **eis**:

  Specify this type to set credentials for connecting to an external Enterprise Information System (EIS), such as SAP, Siebel, JD Edwards, or PeopleSoft.

  You can set the **username** and **password** properties for connecting to an EIS.
- **email**:

  Specify this type to set credentials for connecting to an email server.

  You can set the **username** and **password** properties for connecting to an email server.
- **ftp**:

  Specify this type to set credentials for connecting to an FTP server.

  You can set the **username** and **password** properties for connecting to an FTP server.
- **http**:

  Specify this type to set credentials for static ID identity propagation with SOAP or HTTP request nodes when using basic authentication (basicAuth).

  You can set the **username** and **password** properties for SOAP or HTTP request nodes (SOAPRequest, SOAPAsyncRequest, HTTPRequest, and HTTPAsyncRequest nodes).
- **httpproxy**:

  Specify this type to set credentials for connecting to a secured HTTP proxy server.

  You can set the **username** and **password** properties for connecting to an HTTP server.
- **ims**:

  Specify this type to set credentials for connecting from an IMSRequest node to the IMS server.

  You can set the **username** and **password** properties for connecting to an IMS server.
- **jdbc**:

  Specify this type to set credentials for a JDBC type 4 connection.

  You can set the **username** and **password** properties for connecting to a JDBC resource.
- **jms**:

  Specify this type to set credentials for connecting to JMS resource.

  You can set the **username** and **password** properties for connecting to a JMS resource.
- **jndi**:

  Specify this type to set credentials for connecting to a JNDI resource.

  You can set the **username** and **password** properties for connecting to a JNDI resource.
- **kafka**:

  Specify this type to set credentials for connecting to a secured Kafka cluster.

  You can set the **username** and **password** properties for connecting to a Kafka cluster.
- **kerberos**:

  Specify this type to set credentials for connecting to the Kerberos Key Distribution Center (KDC).

  You can set the **username** and **password** properties for connecting to a Kerberos KDC.
- **keystore**:

  Specify this type to set credentials for opening the web user interface keystore.

  You can set the **password** property for opening the web user interface keystore.

- **keystorekey**:

  Specify this type to set credentials for opening a key inside the web user interface keystore.

  You can set the **password** property for opening the key inside the keystore (for use when the key inside the keystore is protected by a password that is different from the password used to open the keystore).
- **ldap**:

  Specify this type to set Lightweight Directory Access Protocol (LDAP) bind credentials.

  You can set the **username** and **password** properties for binding to an LDAP server.
- **loopback**:

  Specify this type to set credentials for a connection that is made through a LoopBack® connector.

  You can set either the **username** and **password** or the **username**, **password**, **clientId**, and **clientSecret** properties for connecting through a LoopBack connector.
- **mq**:

  Specify this type to set credentials for connecting to a secured IBM MQ queue manager.

  You can set the **username** and **password** properties for connecting to an IBM MQ queue manager.
- **mqtt**:

  Specify this type to set credentials for connecting to a secured external MQTT server, which the integration server uses to publish its event messages.

  You can set the **username** and **password** properties for connecting to an external MQTT server.
- **odbc**:

  Specify this type to set credentials for an Open Database Connectivity (ODBC) data source name (DSN) that is accessed from a message flow.

  You can set the **username** and **password** properties for accessing an ODBC DSN from a message flow.
- **odm**:

  Specify this type to set credentials for an Operational Decision Manager (ODM) Rule Execution Server that is accessed from a message flow.

  You can set the **username** and **password** properties for accessing an ODM Rule Execution Server from a message flow.
- **rest**:

  Specify this type to set credentials for authenticating a connection to an external REST API.

  You can set the following properties for connecting to an external REST API:

  - **apiKey**
  - **username** and **password**
  - **username**, **password**, and **apiKey**
- **salesforce**:

  Specify this type to set credentials for authenticating a connection to a Salesforce system.

  You can set the **username**, **password**, **clientId**, and **clientSecret** properties for accessing a Salesforce system.
- **sftp**:

  Specify this type to set credentials for authenticating a connection to an SFTP server.

  To access an SFTP server, you must specify either the **password** or **sshIdentityFile** property, but not both. If you specify an identity file, you can also specify a passphrase with the **passphrase** property.

- **smtp**:

  Specify this type to set credentials for authenticating a connection to an SMTP server.

  You can set the **username** and **password** properties for connecting to an SMTP server.
- **soap**:

  Specify this type to set credentials for static ID identity propagation with SOAP request and reply nodes when using WS-Security while connecting to or replying from a web service (SOAPRequest, SOAPAsyncRequest, and SOAPReply nodes).

  You can set the **username** and **password** properties to specify the credentials for these connections.
- **truststore**:

  Specify this type to set credentials for connecting to an integration server truststore.

  You can set the **password** property for connecting to a truststore.
- **truststorekey**:

  Specify this type to set credentials for opening a key inside the integration server truststore.

  You can set the **password** property for opening the key inside the truststore (for use when the key inside the truststore is protected by a password that is different from the password used to open the truststore).
- **wsrr**:

  Specify this type to set credentials for connecting to a WebSphere Service Registry and Repository

  You can set the **username** and **password** properties for accessing a WebSphere Service Registry and Repository.
- **wxs**:

  Specify this type to set credentials for connecting to a secure WebSphere eXtreme Scale grid.

  You can set the **username** and **password** properties for accessing a WebSphere eXtreme Scale grid.

For more information about configuring an integration server, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Configuring a default application for an integration server

You can configure a default application for an integration server by setting the application name in the integration server's `server.conf.yaml` configuration file.

Before you begin

- Create an integration server, as described in "Creating an integration server" on page 168.
- Configure the integration server, by following the instructions in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

About this task

All resources that are run by the IBM App Connect Enterprise run time must be located in an application. Any independent resources (located in an independent resource project), are automatically placed in the integration server's default application. You can specify the default application when you create the integration server, by specifying the **--default-application-name** parameter on the command. Alternatively, you can specify a default application for an existing integration server, by setting the **defaultApplication** property in the integration server's `server.conf.yaml` configuration file. When an integration server has been configured to use a default application, independent resources are placed under the specified default application so that they can be accessed.

Procedure

To configure a default application for an integration server, complete the following steps:

1. Use a YAML editor to open the `server.conf.yaml` file.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your integration server configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.

2. Set the **`defaultApplication`** property in the `Defaults` section of the `server.conf.yaml` file, by removing the comment (*#*) from the start of the `defaultApplication` line and specifying a name for the default application that will be used by the integration server:

   ```
   Defaults:
     defaultApplication: 'myDefaultApp'   # Name a default application under which independent
    resources will be placed
   ```

3. Restart the integration server. The properties that you set in the `server.conf.yaml` file take effect when the integration server is started. If you modify these properties again, you must also start the integration server again for the subsequent changes to take effect. For more information, see "Starting an integration server" on page 250.

Configuring the environment to access external applications and resources

If you are developing message flows that access external resources such as databases, Enterprise Information Systems, IMS, or email servers, you must set up your environment to support these applications.

About this task

The topics in this section describe how to configure your environment to support specific applications.

Note: You can also use policies to control certain aspects of message flow and message flow node behavior at run time. For more information, see "Overriding properties at run time with policies" on page 324.

Configuring databases

Configure databases to hold application or business data that you can access from your message flows.

About this task

Databases that hold application or business data can be read from and written to by nodes within the message flows that you deploy to one or more integration nodes in your domain.

For more information about the requirement for, and set up of, databases, and the restrictions that apply, see "Databases overview" on page 1131.

To make databases available, complete the following steps:

Procedure

1. If you want to access databases from your deployed message flows, create and configure the databases and the connections to them.

2. On distributed platforms, create and configure connections to the databases that you have created:
   a) If your message flows use an ODBC connection to a database, enable an ODBC connection for that database.

      Repeat this step for each database that you want to access in this way.

      Note that you can use the **mqsicvp** command as an ODBC test tool; see "Enabling ODBC connections to the databases" on page 184 for further information.
   b) If your message flows use a JDBC connection to a database, enable a JDBC connection for that database.

      Repeat this step for each database that you want to access in this way.

Securing database connections

Set up security for a database connection, whether it is required by the database provider or optional.

Before you begin

- If you are using a JDBC provider, you must first set up the JDBC provider. See Set up your JDBC provider definition.
- If you are using an ODBC connection, you must first connect to the database. See "Enabling ODBC connections to the databases" on page 184.

About this task

Some databases require that all access is associated with a known user ID, for others this association is optional. For example, DB2 requires a data source login name and password on all connections.

Use the **mqsisetdbparms** command to specify a user ID and password that the integration node can use to access each database. If a user ID and password have not been specified for database access using the mqsisetdbparms command, default values for user ID and password are used which are platform specific:

1. Windows On Windows: The integration node service ID and password that you specified on the **mqsicreatebroker** command.
2. Linux UNIX On Linux and UNIX: The user ID `mqsiUser` and password `********` (these values are fixed).

Steps for setting up security are specific to the type of database connection that you are using. Choose your database connection type to see the steps:

- "ODBC connections" on page 183
- "JDBC connections" on page 184

ODBC connections

About this task

If your ODBC data source requires you to define secure access, or if you want to implement security where this is optional, complete the following steps:

Procedure

1. Identify the user IDs that you want to associate with the database connection, or create a user ID with a password, following the appropriate instructions for your operating system and database.
2. Define the user IDs and passwords that the integration node can use to access a particular data source.

3. Run the **mqsisetdbparms** command to create user IDs and passwords that can be used to access the data source from an integration node.

   Use the following format:

   ```
   mqsisetdbparms broker_name -n data_source_name -u database_userID -p database_userID_password
   ```

4. Optional: If you want to use the same user ID and password for more than one database as a default set of credentials, you can specify `dsn::`DSN in the **-n** parameter, as shown in the following example:

   ```
   mqsisetdbparms broker_name -n dsn::DSN -u default_userID -p default_password
   ```

Results

You have secured access to your ODBC data source.

JDBC connections

About this task

If your database requires you to define secure access, or if you want to implement security where this is optional, complete the following steps:

Procedure

1. Identify the user ID that you want to associate with the database connection, or create a user ID with a password, following the appropriate instructions for your operating system and database.
2. Define the user IDs and passwords that the integration node can use for the JDBC connections:
   - Use the following command format:

     ```
     mqsisetdbparms broker_name -n jdbc::security_identity -u userID -p password
     ```

     For example, if you want a user ID *myuserid* with a password of *secretpw* to access a database on integration node *INODE1*, run the following command:

     ```
     mqsisetdbparms INODE1 -n jdbc::mySecurityIdentity -u myuserid -p secretpw
     ```

     The `jdbc::` prefix indicates that the **security_identity** is to be used for JDBC connections.
   - If you want to use the same user ID and password for more than one database as a default set of credentials, you can specify `dsn::`DSN in the **-n** parameter, as shown in the following example:

     ```
     mqsisetdbparms INODE1 -n jdbc::JDBC -u default_userID -p default default_password
     ```

   - Update the corresponding `Security identity` property for the JDBC Providers policy to associate the connection with the security identity that you have defined (see JDBC Providers policy (JDBCProviders)).

Results

You have secured access to your JDBC databases. If you need to define user credentials that can be shared across a business area or account, you can reuse the same security identity that you defined in the previous steps in different JDBC Providers policies.

Enabling ODBC connections to the databases

Set up the resources and environment that the integration node requires for Open Database Connectivity (ODBC) connections to databases on distributed systems.

Before you begin

Read the following topic: “Configuring databases” on page 182.

Note: In IBM App Connect Enterprise, you do not need to install the IBM Integration ODBC Database Extender program to use database nodes in your applications. The program code is installed as part of the IBM App Connect Enterprise installation.

About this task

You can configure both ODBC and Java Database Connectivity (JDBC) connections for access to databases:

- To set up ODBC connections to databases, follow the instructions in this section.

  Optionally, after configuring the ODBC connection parameters, run the command to verify that the integration node can connect to the data source, and to provide useful information about the data source and its interface.On Linux and UNIX systems, this command also checks that the ODBC environment is set up correctly.
- On Linux and UNIX systems, IBM Integration ODBC Database Extender, which encapsulates unixODBC, is provided as the driver manager. IBM App Connect Enterprise provides DataDirect ODBC drivers for interfacing with Oracle, Sybase, and SQLServer databases. For other databases, you must provide the appropriate ODBC driver.

  On Windows, the ODBC driver manager is provided by the operating system. IBM App Connect Enterprise provides DataDirect ODBC drivers for interfacing with Oracle and Sybase. For other databases, you must provide the appropriate ODBC driver.
- To set up JDBC connections to databases, see "Enabling JDBC connections to the databases" on page 200.
- On Linux and UNIX systems, delete the ODBCINI64 environment variable if it exists; it is not required by IBM Integration Bus 10.0. For more information, see "Database connections" on page 1131. The sample `odbc.ini` and `odbcinst.ini` files that are supplied, and the information that is contained in these configuration topics, include all the connection parameters that are supported for connections to your databases. Any additional parameters that are provided by your chosen database drivers are not tested or supported in an IBM App Connect Enterprise environment; consider your requirements carefully before specifying other parameters in your tailored ODBC `.ini` files.

To enable connections on distributed systems:

Procedure

Define the ODBC DSNs according to your platform:

Windows **On Windows:**
Follow the instructions in "Connecting to a database from Windows systems" on page 185.

Linux UNIX **On Linux and UNIX systems:**
For all supported databases, follow the instructions in "Connecting to a database from Linux and UNIX systems by using the IBM Integration ODBC Database Extender" on page 189.

You have now configured the ODBC DSNs for your databases.

Results

You have now enabled the integration node to make connections to your databases.

Connecting to a database from Windows systems

To enable an integration node to connect to a database, define the ODBC data source name (DSN) for the database.

Before you begin

Check that you have set up your environment so that the integration node can connect to the database. Most database managers set up the required environment when you install, but others supply a database

profile that you must run. For information about environments and running database profiles, see "Running database setup scripts before starting an integration node" on page 83.

About this task

Configure an ODBC data source by using the ODBC Data Source Administrator:

1. Click **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.
2. Click the **System DSN** tab and click **Add**.
3. Complete the steps in the following sections for the databases that you are working with.

   If you need more information about a particular database product, see the product-specific documentation.

DB2 UDB

Define a data source for DB2 UDB:

1. Select the driver `IBM DB2 ODBC DRIVER`.
2. Enter the data source name (DSN) and description.
3. Select the correct database alias from the list.
4. Click **Finish** to save your definition.
5. Click **OK** to close the ODBC Data Source Administrator.
6. If you need to use Global Coordination with your database from IBM App Connect Enterprise on Windows systems, the next task is to set up the 32-bit environment that is needed by IBM MQ, see "Setting your environment to support access to databases" on page 200.

You must register the data source as a system data source.

If you prefer, you can use the Configuration Assistant instead of the ODBC Data Source Administrator:

1. Open the DB2 Configuration Assistant.
2. Right-click the database and select **Change Database**.
3. Select **Data Source**.
4. Select **Register this database for ODBC**. Select the system data source option.
5. Click **Finish**.
6. The **Test Connection** dialog opens automatically and you can test the various connections.

Informix® Dynamic Server

Define a data source for Informix Dynamic Server:

1. Select the driver `IBM INFORMIX ODBC DRIVER`.
2. On the **Connection** tab, specify:
   - The Informix server name.
   - The server host name.
   - The Informix network service name (as defined in the services file).
   - The network protocol (for example, `olsoctcp`).
   - The Informix data source name.
   - The user identifier to access the data source within.
   - The password for that user identifier.
3. Click **Apply**.
4. Click **Test Connection** to check your supplied values.
5. Click **OK** to close the ODBC Data Source Administrator.

Microsoft SQL Server

Define a data source for Microsoft SQL Server:

1. Select the driver for the version of SQL Server that you are using:
   - `SQL Native Client` 10.0 for SQL Server 2008.
   - `SQL Native Client` 11.0 for SQL Server 2012, 2014, and 2016.
2. Specify a name and description.
3. Select the correct server from the list.
4. To specify the authentication mode that is used by the server:
   a. Click **Next**.
   b. Select the authentication mode.
   c. Click **Back** to move back to the first panel.
5. Click **Finish** to save your definition.
6. Click **OK** to close the ODBC Data Source Administrator.

Oracle

Define a data source for Oracle:

1. • Select the driver `IBM App Connect Enterprise (10.0.0.`*n*`) DataDirect Technologies 64-BIT Oracle Wire Protocol` where *n* is the level of the installed fix pack.

   The **ODBC Oracle Driver Setup** dialog box opens.
2. On the **General** tab:
   a. Enter the DSN name, description, and host name of the machine where Oracle is running, the port number on which Oracle is listening, and the Oracle Service Name that you want to connect to.
3. On the **Advanced** tab:
   a. Select **Enable SQLDescribeParam**.
   b. Select **Procedure Returns Results**. The resultant ODBC definition in the Windows registry has a string value that is called **`ProcedureRetResults`** with the value 1.
   c. Select **Login Timeout** and set the value to `0`.
   d. Ensure that **Enable N-CHAR** is not selected.
   e. If you are using TIMESTAMP WITH TIMEZONE columns, select **Enable Timestamp With Timezone**.
   f. In **Extended Options**, enter:

```
WorkArounds=536870912;
```

Oracle using Secure Socket Layer (SSL) authentication

Complete the steps for Oracle above.

Then, complete these additional steps:

1. Reopen the **ODBC Oracle Driver Setup** dialog box, see Step 1 for Oracle above.

2. On the **Security** tab:
   a. In the **Authentication section**, set the **Authentication Method** to `Encrypt Password`.
   b. In the **Encryption Section**, set the **Encryption Method** to `SSL Auto`.
   c. Select the check box if you want to **validate the Server certificate**.
   d. Enter a fully qualified path for your **Trust Store**.
   e. Enter your **Trust Store Password**.
   f. Enter a fully qualified path for your **Key Store**.
   g. Enter your **Key Store Password**.
   h. Enter your **SSL Key Password**.
3. Click **OK** to close the ODBC Data Source Administrator.

Oracle using Advanced Security (OAS)

Complete the steps for Oracle above.

Then, complete these additional steps:

1. Reopen the **ODBC Oracle Driver Setup** dialog box, see Step 1 for Oracle above.
2. On the **Advanced Security** tab:
   a. Select the Encryption Level that you want to use. Choose from the following options:
      - 0 - Rejected. If rejected, or no match is found between the driver and server encryption types, data that is sent between the driver and the database server is not encrypted or decrypted. If the Oracle server has its **`sqlnet.encryption_server`** setting set to "REQUIRED" and this option is selected, then the connection to the Oracle database fails.
      - 1 - Accepted. Encryption is used on data that is sent between the driver and the database server if the database server requests or requires it.
      - 2 - Requested. Data that is sent between the driver and the database server is encrypted and decrypted if the database server permits it.
      - 3 - Required. Data that is sent between the driver and the database server must be encrypted and decrypted. If the Oracle server has its **`sqlnet.encryption_server`** setting set to "REJECTED" and this option is selected, then the connection to the Oracle database fails.
   b. Select the Encryption Types that you want to use.
   c. Select the Data Integrity Level you want to use. Choose from the following options:
      - 0 - Rejected. A data integrity check on data that is sent between the driver and the database server is refused. If the Oracle server has its **`sqlnet.crypto_checksum`** setting set to "REQUIRED" and this option is selected, then the connection to the Oracle database fails.
      - 1 - Accepted. A data integrity check can be made on data that is sent between the driver and the database server. Data integrity is used if the database server requests or requires it.
      - 2 - Requested. The driver enables a data integrity check on data that is sent between the driver and the database server if the database server permits it.
      - 3 - Required. A data integrity check must be performed on data that is sent between the driver and the database server. If the Oracle server has its **`sqlnet.crypto_checksum`** setting set to "REJECTED" and this option is selected, then the connection to the Oracle database fails.
   d. Select the Data Integrity Types that you want to use.
3. Click **OK** to close the ODBC Data Source Administrator.

Sybase Adaptive Server Enterprise

Define a data source for Sybase Adaptive Server Enterprise:

1. • Select the driver `IBM App Connect Enterprise (10.0.0.n) DataDirect Technologies 64-BIT Sybase Wire Protocol` where *n* is the level of the installed fix pack.
2. Enter the DSN name, description, and network address of the server, where the network address is made up of **`MyHostMachineName,MyHostMachinePortNumber`**.
3. On the **Advanced** tab:
   - Select **Enable Describe Parameter**.
   - Select **Login Timeout** and set the value to `0`.
   - In **Extended Options**, enter:

     ```
     TimestampTruncationBehavior=1;EnableSPColumnTypes=2;XAConnOptBehavior=3;
     ```
4. On the **Performance** tab:
   - Ensure that the **Prepare Method** setting is `1 - Partial`.

solidDB

Define a data source for solidDB:

1. Select the driver IBM solidDB - (Unicode) DRIVER.
2. Enter the description.
3. Enter the communication port in the network location field, for example, `tcp 2315`.
4. Click **Finish** to save your definition.
5. Click **OK** to close the ODBC Data Source Administrator.

Results

You have now configured your ODBC data source names on Windows.

Connecting to a database from Linux and UNIX systems by using the IBM Integration ODBC Database Extender

IBM Integration ODBC Database Extender encapsulates the unixODBC driver manager. You must set up and configure the integration node to use it.

Before you begin

The following information applies to all supported databases.

- Read the information about the unixODBC Project.
- Create the database.
- Ensure that your database is set up so that the integration node is authorized to access the database.
- Check that you have set up your environment so that the integration node can access the database. You might have to run a database profile that is supplied by the database vendor. For more information, see “Running database setup scripts before starting an integration node” on page 83.

You do not need to install the IBM Integration ODBC Database Extender program to use database nodes in your applications. The program code is installed as part of the IBM App Connect Enterprise installation.

Procedure

1. Copy the `odbc.ini` sample file that is supplied in the *`install_dir`*`/server/ODBC/unixodbc/` directory to a location of your choice.

   Each integration node service user ID on the system can therefore use its own data source name (DSN) definitions.

   Note: To prevent problems with the backup and restore procedures, store the copy of the sample file in the `/var/mqsi` directory, rather than the home directory for your user ID.

2. Ensure that your copy of the `odbc.ini` file is owned by the `mqbrkrs` group, and has 664 permissions.
3. Set the ODBCINI environment variable to point to your copy of the `odbc.ini` file, specifying a full path and file name. Ensure that you point to the copy of the file; do not point to the `odbc.ini` file in the installation directory.
4. Copy the `odbcinst.ini` sample file that is supplied in the *`install_dir`*`/server/ODBC/unixodbc/` directory to a location of your choice. See the **Note** in step 1.
5. Ensure that your copy of the `odbcinst.ini` file is owned by the `mqbrkrs` group and has 664 permissions.
6. Set the ODBCSYSINI environment variable to point to the directory that contains your copy of the `odbcinst.ini` file, specifying a full path name. Ensure that you point to the directory containing the copy; do not point to the directory containing the `odbcinst.ini` file in the installation directory.
7. If you are using the Data Server driver for ODBC that is supplied as part of App Connect Enterprise:
   a) Copy the `db2cli.ini` sample file that is supplied in the *`install_dir`*`/server/ODBC/unixodbc/` directory to a location of your choice.
   b) Ensure that your copy of the `db2cli.ini` file is owned by the `mqbrkrs` group, and that it has 664 permissions.
   c) Set the DB2CLIINIPATH environment variable to point to the directory that contains your copy of the `db2cli.ini file`, specifying a full path name. Ensure that you point to the directory containing the copy; do not point to the directory containing the `db2cli.ini` file in the installation directory.
8. If you are connecting directly to DB2 server, solidDB, or Informix databases, set the library search path environment variable to show the location of the libraries for the database manager that you are using. This step is not required if you are using the Data Server driver for ODBC that is supplied as part of App Connect Enterprise.

   For more information about the library search path, ask your database administrator (DBA), or see the documentation for your database manager.

   The library search path environment variable depends on your platform:

   - Linux On Linux, set `LD_LIBRARY_PATH`.
   - AIX On AIX, set `LIBPATH`.

   Updates to the library search path are not required for other supported databases.
9. If you are using a DB2 database instance that is installed on AIX, a single process can make a maximum of 10 connections that use shared memory to a DB2 database. Use TCP/IP mode to connect to the database instance.
10. Edit the final stanza in the `odbc.ini` file (the [ODBC] stanza), to specify the location of the installed DataDirect ODBC drivers.

    To ensure that you edit the correct `odbc.ini` file, you can open the file in the vi text editor by using the following command:

    ```
    vi $ODBCINI
    ```

    a) In the **`InstallDir`** field, add the IBM App Connect Enterprise installation location to complete the fully qualified path to the ODBC directory.

    Ensure that the path points to the ODBC directory in the IBM App Connect Enterprise installation location. If you do not specify this value correctly, the ODBC definition will not work.

    b) Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

    For example, on AIX:

    ```
    ;##################################################
    ;###### Mandatory information stanza ######
    ```

```
;##############################################

[ODBC]
InstallDir=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers
UseCursorLib=0
IANAAppCodePage=4
UNICODE=UTF-8
```

11. Edit the first stanza in the `odbc.ini` file (the [ODBC Data Sources] stanza) to list the DSN of each database.

    For example, on AIX:

    ```
    ;##############################################
    ;###### List of data sources stanza #######
    ;##############################################
    [ODBC Data Sources]
    DB2DB=IBM DB2 ODBC Driver
    DB2DSDB=IBM DB2 using the Data Server driver included in ACE
    ORACLEDB=DataDirect ODBC Oracle Wire Protocol
    ORACLERACDB=DataDirect ODBC Oracle RAC Wire Protocol
    ORACLESSLDB=DataDirect ODBC Oracle SSL Wire Protocol
    SYBASEDB=DataDirect ODBC Sybase Wire Protocol
    SYBASEDBUTF8=DataDirect ODBC Sybase UTF8 Wire Protocol
    SQLSERVERDB=DataDirect ODBC SQL Server Wire Protocol
    INFORMIXDB=IBM Informix ODBC Driver
    SOLIDDB_DB=IBM Solid DB ODBC Driver
    ```

    List all your DSNs in your `odbc.ini` file, regardless of the database manager. You can define multiple DSNs to resolve to the same database; however, if you are using global coordination of transactions with an Oracle database, do not use this option because it might cause data integrity problems.

12. For each database that you listed in the [ODBC Data Sources] stanza in the `odbc.ini` file, create a data source stanza in the `odbc.ini` file. The entries in the stanza depend on the database manager.

    For a DB2 database instance:

    a. In the **Driver** field:

    - If you are directly connecting to a DB2 server, add the full path of your DB2 installation.
    - If you are using the Data Server driver for ODBC supplied with App Connect Enterprise, ensure that the path points to the driver file in the App Connect Enterprise installation location.

    b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

    c. In the **Database** field, type the DB2 alias. The data source name must be the same as the database alias name. If you are using a remote DB2 database, you must set up your client/server connection to resolve this alias to the correct database. For more information, see the DB2 documentation.

    If the requirement is to have multiple stanzas that refer to the same DB2 database, aliases must be created in DB2 by using the DB2 CATALOG command. These aliases can then have their own stanza in the `odbc.ini` file.

    The `odbc.ini` file cannot be used to set up aliases for DB2.

    For example, connecting directly to a DB2 server on AIX:

    ```
    ;# DB2 stanza
    [MYDB2DB]
    DRIVER=/opt/IBM/db2/V11.1/lib64/db2o.o
    Description=IBM DB2 ODBC Database
    Database=MYDB2DB
    ```

    Connecting directly to a DB2 server on Linux systems:

    ```
    ;# DB2 stanza
    [MYDB2DB]
    DRIVER=/opt/IBM/db2/V11.1/lib64/libdb2o.so
    Description=IBM DB2 ODBC Database
    ```

```
Database=MYDB2DB
```

Connecting to DB2 using the Data Server Driver on Linux Systems:

```
;# DB2 using DataServer driver included in ACE
[MYDB2DB]
DRIVER=/usr/opt/IBM/mqsi/11.0.0.11/server/ODBC/dsdriver/odbc_cli/clidriver/lib/
libdb2o.so
Description=IBM DB2 ODBC Database
Database=MYDB2DB
```

For an Oracle database:

For all platforms:

a. In the **Driver** field, ensure that the path points to the driver file in the IBM App Connect Enterprise installation location, as shown in the following example.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **HostName** field, type the name or IP address of the machine that is hosting your Oracle system.

d. In the **PortNumber** field, type the number of the port on which your Oracle server is listening on the machine that you specified in the **HostName** field.

e. In the **ServiceName**field, type the Oracle service name that you want to connect to on the system you specified in the **HostName** field.

f. If you are using TIMESTAMP WITH TIMEZONE columns, uncomment the `EnableTimestampwithTimezone` setting.

g. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example, on AIX:

```
;# Oracle stanza
[MYORACLEDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKora95.so
Description=DataDirect ODBC Oracle Wire Protocol
HostName=my-machine.hursley.ibm.com
PortNumber=1521
ServiceName=my-oracle-service
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
EnableNcharSupport=0
;# Un-comment the next setting if you wish to use Oracle TIMESTAMP WITH TIMEZONE
 columns
```

```
;# EnableTimestampwithTimezone=1
```

For an Oracle database that uses Real Application Clusters:

For all platforms:

a. In the **Driver** field, ensure that the path points to the driver file in the IBM App Connect Enterprise installation location, as shown in the following example.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **HostName** field, type the name or IP address of the machine that is hosting your primary (preferred) Oracle instance.

d. In the **PortNumber** field, type the number of the port on which your Oracle server is listening on the machine that you specified in the **HostName** field.

e. In the **ServiceName** field, type the Oracle Real Application Cluster service name that you want to connect to on the system that you specified in the **HostName** field.

f. If you are using TIMESTAMP WITH TIMEZONE columns, uncomment the `EnableTimestampwithTimezone` setting.

g. In the **AlternateServers** field, provide a list of alternative locations for this service for situations when the primary location, which is defined in **HostName**, is unavailable. Each location specification consists of three parts, which are separated by colons. Enter these values as one continuous string; the text in this example has been split to improve readability.

```
HostName=<Alternative host name>
:PortNumber=<Oracle listner port on alternative server>
:ServiceName=<Service name on the alternative server>
```

If you want to specify more than one AlternateServer, separate each additional location specification with a comma. Whenever a new database connection is required, for example after an Oracle instance failover, the primary location will be tried first. However, if the primary location is unavailable, the driver will try the list of alternative locations in turn.

h. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example, on AIX:

```
;# Oracle Real Application Clusters stanza
[MYORACLERACDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKora95.so
Description=DataDirect ODBC Oracle RAC Wire Protocol
HostName=my-primary-machine.hursley.ibm.com
PortNumber=1521
ServiceName=my-oracle-rac-service
;#This shows one alternate server definition. Add extra ones using a ',' to seperate
 each definition.
AlternateServers=(HostName=my-first-backup-
machine.hursley.ibm.com:PortNumber=1521:ServiceName=my-
oracle-rac-first-backup-service,HostName=my-second-backup-
machine.hursley.ibm.com:PortNumber=1521:ServiceName=my-oracle-rac-second-backup-
service)
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
EnableNcharSupport=0
;# Un-comment the next setting if you wish to use Oracle TIMESTAMP WITH TIMEZONE
 columns
```

```
;# EnableTimestampwithTimezone=1
```

For an Oracle database that uses Secure Socket Layer (SSL):

For all platforms:

a. In the **Driver** field, ensure that the path points to the driver file in the IBM App Connect Enterprise installation location, as shown in the following example.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **HostName** field, type the name or IP address of the machine that is hosting your primary (preferred) Oracle instance.

d. In the **PortNumber** field, type the number of the port on which your Oracle server is listening for SSL connections on the machine you specified in **HostName**.

e. In the **ServiceName** field, type the Oracle SSL service name that you want to connect to on the system you specified in the **HostName** field.

f. If you are using TIMESTAMP WITH TIMEZONE columns, uncomment the `EnableTimestampwithTimezone` setting.

g. In the **KeyPassword** field, type your SSL key password.

h. In the **KeyStore** field, type the fully qualified name of your SSL key store.

i. In the **KeyStorePassword** field, type your SSL key store password.

j. In the **TrustStore** field, type the fully qualified name of your SSL trust store.

k. In the **TrustStorePassword** field, type your SSL trust store password.

l. In the **EncryptionMethod** field, type the method to use to encrypt data sent between the driver and the database server. Valid values are as follows:

- **0** No encryption. This value is the default.
- **1** SSL. If the server supports protocol negotiation, the driver and server negotiate the use of the SSL protocols specified in the **CryptoProtocolVersion** connection option.
- **3** SSL3.
- **4** SSL2.
- **5** TLS1.

m. If the **EncryptionMethod** has been set to 1, use **CryptoProtocolVersion** to specify a comma-separated list of the cryptographic protocols to use. Valid protocols are TLSv1.2, TLSv1.1, TLSv1, SSLv3 and SSLv2. When multiple protocols are specified, the driver uses the highest version supported by the server. The default value is `TLSv1.2, TLSv1.1, TLSv1`.

n. In the **ValidateServerCertificate** field, type 1 to enable validation of the certificate that is sent by the database server when SSL encryption is enabled.

o. Accept the default values shown in the sample `odbc.ini` file for all the other entries in the stanza.

For example, on AIX:

```
;# Oracle using SSL stanza
[MYORACLESSLDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKora95.so
Description=DataDirect ODBC Oracle Wire Protocol
HostName=my-machine.hursley.ibm.com
PortNumber=2484
ServiceName=my-oracle-ssl-service
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
```

```
AuthenticationMethod=1
KeyPassword=my-password
KeyStore=/Development/ssl/my-store.p12
KeyStorePassword=my-password
TrustStore=/Development/ssl/my-store.p12
TrustStorePassword=my-password
EncryptionMethod=1
CryptoProtocolVersion=TLSv1.2
ValidateServerCertificate=1
EnableNcharSupport=0
;# Un-comment the next setting if you wish to use Oracle TIMESTAMP WITH TIMEZONE
 columns
;# EnableTimestampwithTimezone=1
```

For an Oracle database that uses Advanced Security (OAS):

For all platforms:

a. In the **Driver** field, ensure that the path points to the driver file in the IBM App Connect Enterprise installation location, as shown in the following example.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **HostName** field, type the name or IP address of the machine that is hosting your Oracle system.

d. In the **PortNumber** field, type the number of the port on which your Oracle server is listening on the machine you specified in **HostName**.

e. In the **ServiceName** field, type the Oracle service name that you want to connect to on the system that you specified in the **HostName** field.

f. If you are using TIMESTAMP WITH TIMEZONE columns, uncomment the `EnableTimestampwithTimezone` setting.

g. In the **EncryptionLevel** field, enter the level of encryption that you are using. Choose one value from the following options:

- 0 - Rejected. If rejected, or no match is found between the driver and server encryption types, data that is sent between the driver and the database server is not encrypted or decrypted. If the Oracle server has its **sqlnet.encryption_server** setting set to "REQUIRED" and this option is selected, then the connection to the Oracle database fails.
- 1 - Accepted. Encryption is used on data that is sent between the driver and the database server if the database server requests or requires it.
- 2 - Requested. Data that is sent between the driver and the database server is encrypted and decrypted if the database server permits it.
- 3 - Required. Data that is sent between the driver and the database server must be encrypted and decrypted. If the Oracle server has its **sqlnet.encryption_server** setting set to "REJECTED" and this option is selected, then the connection to the Oracle database fails.

h. In the **DataIntegrityLevel** field, choose one value from the following options:

- 0 - Rejected. A data integrity check on data that is sent between the driver and the database server is refused. If the Oracle server has its **sqlnet.crypto_checksum** setting set to "REQUIRED" and this option is selected, then the connection to the Oracle database fails.
- 1 - Accepted. A data integrity check can be made on data that is sent between the driver and the database server. Data integrity is used if the database server requests or requires it.
- 2 - Requested. The driver enables a data integrity check on data that is sent between the driver and the database server if the database server permits it.
- 3 - Required. A data integrity check must be performed on data that is sent between the driver and the database server. If the Oracle server has its **sqlnet.crypto_checksum**

setting set to "REJECTED" and this option is selected, then the connection to the Oracle database fails.

i. Accept the default values shown in the sample odbc.ini file for all the other entries in the stanza.

For example, on AIX:

```
;# Oracle using SSL stanza
[MYORACLEOASDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKora95.so
Description=DataDirect ODBC Oracle Wire Protocol
HostName=my-machine.hursley.ibm.com
PortNumber=1586
ServiceName=my-oracle-oas-service
CatalogOptions=0
EnableStaticCursorsForLongData=0
ApplicationUsingThreads=1
EnableDescribeParam=1
OptimizePrepare=1
WorkArounds=536870912
ProcedureRetResults=1
ColumnSizeAsCharacter=1
LoginTimeout=0
EncryptionTypes=AES128,AES192,AES256,RC4_40,RC4_56,RC4_128,RC4_256,DES,3DES112,3DES168
EncryptionLevel=3
DataIntegrityTypes=SHA1,MD5
DataIntegrityLevel=3
EnableNcharSupport=0
;# Un-comment the next setting if you wish to use Oracle TIMESTAMP WITH TIMEZONE
 columns
;# EnableTimestampwithTimezone=1
```

For a Sybase database:

For all platforms except Linux on Z:

a. In the **Driver** field, ensure that the path points to the driver file in the IBM App Connect Enterprise installation location, as shown in the following example.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **Database** field, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

d. In the **NetworkAddress** field, type the network address of your Sybase ASE server (this address is required for local and remote databases). Specify an IP address or server name, in the following format:

```
Your Sybase server name or IP address,Your Sybase port number
```

For example:

```
Sybaseserver,5000
```

You can also specify the IP address directly; for example:

```
199.226.224.34,5000
```

You can find the port number in the Sybase interfaces file, which is named interfaces.

e. Accept the default values shown in the sample odbc.ini file for all the other entries in the stanza.

For example, on AIX:

```
;# Sybase Stanza
[MYSYBASEDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKase95.so
Description=DataDirect ODBC Sybase Wire Protocol
Database=SYBASEDB1
ApplicationUsingThreads=1
```

```
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=my-machine.hursley.ibm.com:4100
SelectUserName=1
ColumnSizeAsCharacter=1
EnableSPColumnTypes=2
LoginTimeout=0
TimestampTruncationBehavior=1
XAConnOptBehavior=3
```

If you want to use a UNICODE UTF8 Sybase data source, add the following line to the end of your Sybase stanza:

```
Charset=UTF8
```

For remote access to an SQL Server database

For all platforms:

a. In the **Driver** field, add the IBM App Connect Enterprise installation location to complete the fully qualified path to the driver shown in the sample odbc.ini file.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **Database** field, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

d. In the **HostName** field, type the name or IP address of the server to which you want to connect.

e. In the **PortNumber** field, type the number of the port of the server listener.

f. To specify a named instance of SQL Server, replace the **HostName** and **PortNumber** lines with HostName=*Your SQLServer Machine Name**Your SQLServer Instance Name*

g. Accept the default values shown in the sample odbc.ini file for all the other entries in the stanza.

For example, on AIX:

```
;# UNIX to SQLServer stanza
[MYSQLSERVERDB]
Driver=/usr/opt/IBM/mqsi/11.0.0.2/server/ODBC/drivers/lib/UKsqls95.so
Description=DataDirect ODBC SQL Server Wire Protocol
Database=SQLSERVERDB
HostName=my-machine.hursley.ibm.com
PortNumber=1433
AnsiNPW=1
LoginTimeout=0
QueryTimeout=0
;# To specify a named instance of SQL Server replace the HostName and PortNumber
 lines with
;# HostName=<Your SQLServer Machine Name>\<Your SQLServer Instance Name>

;# To use Integrated Windows Authentication, reinstate the following line:
;# AuthenticationMethod=9
```

```
;# Domain=<Your Windows Domain Name>
```

h. If you want to use Integrated Windows Authentication (IWA) for access to the remote SQL Server database, reinstate and complete the lines at the end of the stanza.

For an Informix database:

a. In the **Driver** field, add the full path of your Informix Client library.

b. In the **Description** field, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In the **ServerName** field, type the name of the Informix IDS server.

d. In the **Database** field, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

For example, on AIX:

```
;# Informix Stanza
[MYINFORMIXDB]
Driver=/opt/IBM/informix/lib/cli/iclit09b.so
Description=IBM Informix ODBC Database
ServerName=my-machine
Database=MYDB
```

For a solidDB database:

For all platforms except Linux on POWER and Linux on Z:

Client side

odbc.ini file

a. In **Driver**, add the full path of your solidDB Client library.

b. In **Description**, type a meaningful description of the database. This field is for information only and does not affect the connection.

c. In **Database**, type the name of the database to which you want to connect by default. If you do not specify a value, the default value is the database that is defined by your system administrator for each user.

For example, on AIX:

```
;# SolidDB Stanza
[SOLID_DB]
Driver=/opt/solidDB/bin/soca5x6465.so
Description=IBM Solid DB ODBC database
```

```
Database=SOLIDDB_DB
```

Note: all additional information is ignored.

`solid.ini` file

a. This configuration file is located in the directory that is referenced by the environment variable SOLIDDIR.

b. The `solid.ini` mapping is from the data source name (as defined in `ODBCINI`) to the solidDB connection string.

c. The connection string takes the form <logical name of the driver> = <physical solidDB connect string>.

d. The Physical connection string specifies the:

- Protocol
- Machine name or IP address
- Port number to use

For example, on AIX:

```
[Data Sources]
SOLIDDB_DB=tcp my_aix_system 1964
```

Server side

`solid.ini` file

a. This configuration file is located in the installation directory for solidDB.

b. Set the Data Source as for the Client side `solid.ini` file.

c. Set **`Listen`** to where the listener for the server is located.

d. Set **`CharPadding=yes`** and **`NumericPadding=yes`** to turn on padding.

For example, on AIX:

```
[Data Sources]
SOLIDDB_DB=tcp my_aix_system 1964

[COM]
Listen=tcpip 1964

[SQL]
CharPadding=yes
NumericPadding=yes
```

13. Ensure that you have edited all necessary parts of all of the relevant `.ini` files:

- The [ODBC Data Source] stanza at the top of the `odbc.ini` file.
- A stanza for each data source in the `odbc.ini` file.
- The [ODBC] stanza at the end of the `odbc.ini` file.
- Additionally, for solidDB, both the client and server-side `solid.ini` files.

If you do not configure all parts correctly, the ODBC DSNs do not work and the integration node is unable to connect to the database.

14. Optional: On AIX, database connect times can sometimes take marginally longer than on Linux platforms due to the IBM Integration ODBC Database Extender searching for unicode conversion libraries that do not typically exist on AIX. To prevent this automatic search, you can set the following property in the `[ODBC]` stanza of the `odbcinst.ini` file:

```
IconvEncoding=UCS-2
```

Results

You have now configured database connections on Linux and AIX. You can check that the ODBC environment is configured correctly by running the **mqsicvp** command. For more information, see command.

Setting your environment to support access to databases

When you have configured your ODBC data source names (DSNs), you must also configure the environment so that you can issue console commands, and the integration node that you start can access the required database libraries. For example, if you have a DB2 database, you must add the DB2 client libraries to your library search path.

About this task

Windows

1. On Windows platforms, the environment is typically set up for you when you install the database product, and no further action is required. However, some database managers provide a database profile that you must run to enable the connection from the integration node; for further information, see “Running database setup scripts before starting an integration node” on page 83.

Linux UNIX

1. On Linux and UNIX systems, run a profile for each database you want to access. For example, on DB2 you must run `db2profile`; other database vendors have similar profiles. For further information, see “Running database setup scripts before starting an integration node” on page 83.

Enabling JDBC connections to the databases

Configure connections to a database through a JDBC Providers policy.

About this task

Use a JDBC connection from Java programs that are associated with a JavaCompute node or a user-defined node that is written in Java.

You must also set up JDBC connections if your message flows include graphical data maps with one or more database transforms to be run from a Mapping node, or if they include DatabaseRetrieve or DatabaseRoute nodes.

If you configure a JDBC type 4 connection from an application running on a Linux, UNIX, or Windows system, you can configure your integration node and queue manager to include interactions with the databases in globally-coordinated transactions. On z/OS, JDBC connections can be coordinated only by the integration node.

The information provided in this section is independent of whether your operating systems, integration nodes, integration servers, queue managers, and databases operate in 32-bit or 64-bit mode, except where stated.

When you write Java classes for a JavaCompute node or a user-defined node, your code must comply with the following restrictions:

- Do not include any code that performs a COMMIT or a ROLLBACK function.
- Do not close the connection to the database. The integration node manages all connections, and closes a connection if it is idle for approximately one minute, or if the message flow completes.

To configure JDBC type 4 connections:

Procedure

1. Set up your JDBC provider definition.
2. Optional: Set up security.

3. Optional: If your integration node is running on a Windows system, authorize access to JDBCProvider resources.

What to do next

If you have been following the instructions in "Working with databases" on page 1130 or Mapping database content, the next task is "Setting up a JDBC provider for type 4 connections" on page 201.

When you have completed configuration of the databases, add or modify Java code in your JavaCompute or user-defined nodes to access the database that is identified in the JDBC Providers policy.

Setting up a JDBC provider for type 4 connections

Use a JDBC Providers policy to configure a JDBC provider service.

Before you begin

- See the following information: "Creating an integration node" on page 114.
- Create the database by following the database documentation.

About this task

When you include a DatabaseRetrieve, DatabaseRoute, JavaCompute, Mapping, or Java user-defined node in a message flow, and interact with a database in that node, the integration server must establish a connection with the database to fulfill the operations that are performed by the node. You must define a JDBC Providers policy to provide the integration server with the information that it needs to complete the connection.

Important: When naming your JDBC Providers policy, consider the following requirements:

- If you want to use your JDBC Providers policy with a JavaCompute node, or with a Java user-defined node, the name of your policy must match the *datasourceName* parameter in the `getJDBCType4Connection()`, provided that the policy is in the default policy project. If the policy is not in the default policy project, you must specify the policy and policy project name in the format `{`*`policyProjectName`*`}:`*`PolicyName`*.
- If you want to use your JDBC Providers policy with a Mapping node, the name of your policy must match the database name that is used by the database transforms in your Graphical Data Map. For each database transform, the database name is determined by the database definition (`.dbm` file) in the Data Design project that was used to create the map. The JDBC Providers policy must be deployed in the default policy project; for more information, see "Configuring a default policy project" on page 329.
- If you want to use your JDBC Providers policy with a DatabaseRetrieve node, or with a DatabaseRoute node, the name of your policy must match the value of the *Data source name* property of the node. The JDBC Providers policy must be deployed in the default policy project; for more information, see "Configuring a default policy project" on page 329.

A JDBC Providers policy supports connections to one database only; you must create a policy for each database that your nodes or Java applications connect to.

To set up a JDBC provider for type 4 connections, complete the following steps.

Procedure

1. Identify the type of database for which you require a JDBC Providers policy.

   Supported JDBC drivers and databases are shown in IBM App Connect Enterprise system requirements; support for globally coordinated (XA) transactions is restricted on some platforms and for some databases.
2. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a JDBC Providers policy and choose the template for your chosen database type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).

3. The template provides some default values, but you must change some of them to create a viable definition.
   For example, the `Database name` property is mandatory, but is initially empty (see JDBC Providers policy (JDBCProviders)).

   The following values and order of preference are used by the integration server to substitute the user ID and password in the pattern:

   a. First, on all platforms: The user ID and password that you have set for the specific database, by using the **mqsisetdbparms** and specifying the database in the **-n** parameter.

   b. Second, on all platforms: The user ID and password that you have set for all other databases, by using the **mqsisetdbparms** and specifying `jdbc::JDBC` in the **-n** parameter.

   c. Third, the values are platform-specific:

      i) Windows On Windows: The integration node service ID and password that you specified on the **mqsicreatebroker** command.

      ii) Linux UNIX On Linux and UNIX: The user ID `mqsiUser` and password `********` (these values are fixed).

4. When you have created your JDBC Providers policy, you must deploy it to any integration server where you will deploy the message flows that will use this policy. You can deploy the policy project directly to an integration server.

What to do next

If required, set up security for the JDBC connection, set up the environment to include the JDBC Providers policy in globally coordinated transactions, or both.

Configuring a JDBC type 4 connection for globally coordinated transactions

If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

Before you begin

Set up your JDBC provider definition.

About this task

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

Updates that you make to a database across a JDBC type 4 connection can be coordinated with other actions taken within the message flow, if you set up the resources to support coordination.

Complete the following steps:

Procedure

1. Check that the definition of your JDBC Providers policy is appropriate for coordinated transactions.

   For example, to set up the required JDBC classes:

   - For DB2, set **JDBC type 4 data source class name** to `com.ibm.db2.jcc.DB2XADataSource` and **JDBC driver class name** to `com.ibm.db2.jcc.DB2Driver`.
   - For Oracle, set **JDBC type 4 data source class name** to `oracle.jdbc.xa.client.OracleXADataSource` and **JDBC driver class name** to `oracle.jdbc.OracleDriver`.

   Consult your database administrator or the documentation provided by your database supplier, to confirm that all the JDBC Providers policy properties are set appropriately. For example, a database supplier might require secure access if it is participating in coordinated transactions.

2. Define the switch file and the database properties:

   a) The JDBC switch file is supplied by IBM App Connect Enterprise, and uses static XA registration (see "Configuring databases for global coordination of transactions" on page 704).

   b) Linux

   On Linux and UNIX systems, open the `qm.ini` file for the integration node queue manager with a text editor. Add the following stanza for each database:

   ```
   XAResourceManager:
          Name=Database_Name
          SwitchFile=JDBCSwitch
          XAOpenString=JDBC_DataSource
          ThreadOfControl=THREAD
   ```

   Database_Name is the database name (DSN) of the database defined to the JDBC Providers policy.

   `JDBCSwitch` is a fixed generic name that represents the switch file for XA coordination. Use this value, or another single fixed value, in each stanza; the specific switch file that the queue manager uses is defined by the symbolic links you create in the next step.

   JDBC_DataSource is the identifier of the JDBC Providers policy.

   Define a stanza for each database (DSN) that you connect to from this integration server. You must create separate definitions even if the DSNs resolve to the same physical database. Therefore, you must have a stanza for each JDBC Providers policy that you have defined, because each service can define the properties for a single database.

   c) Windows

   On Windows, open IBM MQ Explorer and select the queue manager, for example BROKERQM.

   Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

   Enter *JDBCSwitch* in **SwitchFile**.

3. Set up queue manager access to the switch file:

   a) Linux

   On Linux and UNIX systems, create a symbolic link to the switch files that are supplied in your *install_dir*`/server/lib` directory.

   install_dir is the directory to which you installed IBM App Connect Enterprise. The default location for this directory is *install_dir*`/ace-12.0.`*n*`.0/server` on Linux, or `/opt/IBM/`

`mqsi/12.0.n.0/server` on UNIX systems. The default directory includes the version, release, modification, and fix of the product, in the format v.r.m.f (version.release.modification.fix).

Set up links in the `/var/mqm/exits64` directory. The file name for all platforms is `libJDBCSwitch.so`.

Specify the same name of the switch file, `JDBCSwitch` or your own value, in the `/exits64` directory. For example, on AIX:

```
ln -s install_dir/server/lib/libJDBCSwitch.so /var/mqm/exits64/JDBCSwitch
```

b) Windows

On Windows, copy the `JDBCSwitch32.dll` file from the *install_dir*`\server\bin` directory to the `\exits` subdirectory in the IBM MQ installation directory, and rename the file to `JDBCSwitch.dll`. Then, copy the `JDBCSwitch.dll` file from the *install_dir*`\server\bin` directory to the `\exits64` subdirectory in the IBM MQ installation directory.

4. Configure the message flow that includes one or more nodes that access databases that are to participate in a globally coordinated transaction.
   a) Open an IBM App Connect Enterprise Toolkit session.
   b) Switch to the Integration Development perspective.
   c) Add the message flow that includes the node or nodes that connect to the database that is to participate in a globally coordinated transaction to a new or existing BAR file.
   d) Build the BAR file.
   e) Click the **Configure** tab, select the message flow that you have added, and select the **Coordinated Transaction** check box.

What to do next

If your integration node is running on Windows, authorize the queue manager to access resources that are associated with the JDBC Providers policy (see "Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 204).

If you have been following the instructions in "Working with databases" on page 1130, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 708 (optional).

Authorizing access to JDBC type 4 JDBCProvider resources on Windows

Authorize the integration node and queue manager to access shared resources that are associated with the JDBCProvider. This task is required only if you want the database updates to be included in globally coordinated transactions on Windows systems.

Before you begin

Set up your JDBC provider definition.

About this task

On distributed systems, an IBM MQ queue manager provides the coordinated transaction support, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the flow. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

When the queue manager coordinates transactions, both queue manager and integration node access shared memory to control a connection to the databases with which the message flow interacts. Therefore, they require the same access control of the shared memory. One method to achieve this control is to use the same ID for the integration node service ID and the queue manager administrative ID.

If you specified an existing queue manager when you created the integration node, check that its administrative ID is the same ID as the one used for the service ID of the integration node. If the ID is not the same, change the queue manager ID to be the same as the integration node service ID.

Complete the following steps on the Windows system on which the integration node is running:

Procedure

1. Click **Start** > **Run** and enter dcomcnfg. The Component Services window opens.
2. In the left pane, expand **Component Services** > **Computers** > **My Computer** and click **DCOM Config**.
3. In the right pane, right-click the IBM MQ service labeled **IBM MQSeries Services**, and click **Properties**.
4. Click the **Identity** tab.
5. Select **This user** and enter the user ID and password for the integration node service ID to associate that ID with the queue manager.
6. Click **OK** to confirm the change.

Enabling data encryption for a JDBC connection

Encryption of JDBC connection is managed by parameters passed to the third party JDBC client jars that are supplied by the JDBC provider. You can use the IBM App Connect Enterprise JDBC Providers policy or a vendor-specific configuration file to pass the parameters.

About this task

Encryption parameters are specific to a JDBC provider. Refer to the documentation issued by your JDBC provider for the details of the Java encryption parameters that you require in your runtime environment.

To enable data encryption for a JDBC connection, follow the instructions in one of the following sections:

- "Using the IBM App Connect Enterprise JDBC Providers policy to enable data encryption for JDBC connections " on page 205
- "Using a vendor-specific configuration file to enable data encryption for JDBC connections " on page 206

Note: You can also use either method to apply extra environment parameters for a JDBC connection; for example, when you configure JDBC with SSL, or with a high availability and scalability feature such as Oracle RAC. SSL setup requires additional steps and these are described in "Setting up a public key infrastructure" on page 2635.

Using the IBM App Connect Enterprise JDBC Providers policy to enable data encryption for JDBC connections

About this task

For information about how policies are used to enable JDBC connections, see "Enabling JDBC connections to the databases" on page 200.

The encryption parameters are set in the **`environmentParms`** property of the JDBC Providers policy; the property applies extra parameters to the JDBC connection URL.

In the example described in this section, the Oracle JDBC thin client is used to explain how to configure IBM App Connect Enterprise to enable the encryption features in the JDBC client JARs. The following parameters are set to enable encryption of data between IBM App Connect Enterprise and an Oracle database:

- ORACLE.NET.ENCRYPTION_CLIENT=REQUIRED
- ORACLE.NET.ENCRYPTION_TYPES_CLIENT(AES256)
- ORACLE.NET.CRYPTO_CHECKSUM_CLIENT=REQUIRED
- ORACLE.NET.CRYPTO_CHECKSUM_TYPES_CLIENT=(SHA256,SHA1)

This configuration method is particularly suitable when there is a limited set of parameters, or when different parameters need to be customized for multiple JDBC Providers policies.

Procedure

Complete the following step:

- Set the encryption parameters as name-value pairs separated by a semicolon by issuing the **mqsichangeproperties** command. For example:

```
mqsichangeproperties integrationNodeName -c JDBCProviders -o Oracle -n environmentParms
-v
 oracle.net.encryption_client=REQUIRED;oracle.net.encryption_types_client=AES256;oracle.net.crypto_checksu
```

Using a vendor-specific configuration file to enable data encryption for JDBC connections

About this task

Alternatively, you can use a vendor-specific configuration file that contains the encryption parameters. The location of this file is specified by a JVM system property that is runtime environment of the integration server. Update the JDBC Providers policy to refer to the relevant part of the configuration file.

The encryption parameters can be set as stanzas in an Oracle configuration file called TNSNAMES.ORA. The location of the configuration file is made available to an integration server by using a Java system property.

Procedure

Complete the following steps:

- Make the location of the configuration file available and issue the **mqsichangeproperties** command to update **serverName**:
  a. Make available the location of the configuration file to the integration server by specifying a Java system property on the **mqsichangeproperties** command. For example:

```
mqsichangeproperties integrationNodeName -e integrationServerName  -o ComIbmJVMManager
-n jvmSystemProperty -v "-Doracle.net.tns_admin=Location of TNSNAMES.ORA file"
```

  b. Issue the **mqsichangeproperties** command to update the **serverName** property of the JDBC Providers policy to specify the name of the Oracle Net service in the TNSNAMES.ORA file. For example:

```
mqsichangeproperties integrationNodeName -c JDBCProviders -o Oracle -n serverName
-v Name of Oracle Net service
```

Support for Unicode and DBCS data in databases

You can manipulate Unicode Standard version 3.0 data in suitably configured databases, using ESQL, in nodes that access databases through ODBC. Support for the manipulation of Unicode data is not available for nodes that access databases through JDBC.

IBM App Connect Enterprise does not support DBCS-only columns in tables that are defined in databases; therefore, the following data types are not supported through ODBC or JDBC connections:

- NCHAR, NVARCHAR, NVARCHAR2, NCLOB (on Oracle)
- NCHAR, NVARCHAR, NTEXT, UNICHAR, UNIVARCHAR (on Sybase)
- NCHAR, NVARCHAR (on Informix)

GRAPHIC, VARGRAPHIC, LONGVARGRAPHIC, and DBCLOB data type support on DB2 is provided for App Connect Enterprise with the following limitations:

- Using the DB2 string functions with Unicode data can return unexpected results. For more information, see "Unicode string functions in DB2" on page 207.

Support for Unicode is available only for the generally-supported versions of the following database managers:

- IBM DB2 for Windows, Linux, UNIX, and z/OS operating systems.
- Oracle
- Microsoft SQL server
- Sybase Adaptive Server Enterprise (ASE)

For information about the versions of databases supported, see IBM App Connect Enterprise system requirements.

Support for the manipulation of Unicode data is not available for nodes that access databases that use JDBC; for example, DatabaseRetrieve and DatabaseRoute.

If you are using DB2:

- On Windows, Linux, and UNIX operating systems, your database must be created with code set `utf-8`.
- On all platforms, DB2 returns the lengths of strings in bytes, rather than characters; this response has implications for the behavior of string length-related ESQL functions.

  Some functions might fail, or function differently, when processed by the database. See "Unicode string functions in DB2" on page 207 for further information.

If you are using Oracle:

- Your database must be created with NLS_CHARACTERSET of AL32UTF8.
- Your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC ini files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See "Enabling ODBC connections to the databases" on page 184 for further information.
- For 32-bit connections, you must set the variable NLS_LANG in the App Connect Enterprise environment to the value `<yourlanguage>_<yourterritory>.AL32UTF8`.

if you are using Microsoft SQL server:

- You must use NCHAR, NVARCHAR, and NTEXT data types for your column definitions.
- For App Connect Enterprise on UNIX and Linux platforms, your ODBC data source definition must include the setting `ColumnSizeAsCharacter=1`; this setting must be included in the appropriate stanza in the ODBC `.ini` files.

If you are using Sybase ASE:

- The default character set of your ASE server must be `UTF-8`.
- Your ODBC data source definition must include the settings `ColumnSizeAsCharacter=1` and `CharSet=UTF8`.

  On UNIX and Linux platforms, this setting must be included in the appropriate stanza in the ODBC `.ini` files.

  On Windows platforms, this string value must be added to the ODBC data source key in the registry.

  See "Enabling ODBC connections to the databases" on page 184 for further information.

Unicode string functions in DB2

When you use string functions in ESQL expressions, certain parameters refer to character positions or counts.

For example, with the SUBSTRING function, coding:

```
SUBSTRING('Hello World!' FROM 7 FOR 4)
```

results in the string `Worl` because 7 refers to the seventh character position, and 4 refers to four characters.

If the string `Hello World!` is represented in an ASCII code page, the seventh *byte* of that representation is the seventh character. However, in other code pages or encoding schemes (for example, some Unicode representations) the seventh character could start at byte 13, and 4 characters can occupy 8 bytes.

IBM App Connect Enterprise correctly handles this situation; that is, the numeric parameters and results of these string functions always refer to *characters* and not the bytes used to represent them.

In some situations, IBM App Connect Enterprise delegates expressions to a database engine for processing. For example, if there is a WHERE clause, in a SELECT function, applied to a database data source, the database is passed the WHERE clause if it can interpret all the functions in the expression.

If there are functions that are not supported by the database, IBM App Connect Enterprise passes only those parts of the expression that can be interpreted, retrieves an unfiltered record set, and performs the remaining filtering itself.

DB2 string functions use *byte* indexing and not *character* indexing. Therefore, for Unicode data, the meaning of certain functions differs from the IBM App Connect Enterprise functions, even though they can be ‘interpreted'.

Characters in Unicode UTF8 representation, for example, can occupy from 1-4 bytes, so that the seventh character can start anywhere from byte 7 to byte 25.

The following string functions are affected:

- INSERT function
- LEFT function
- LENGTH function
- OVERLAY function
- POSITION function
- RIGHT function
- SPACE function
- SUBSTRING function

These functions either take numeric parameters, or return numeric results that refer to indexes, or counts, of characters in a string.

When expressions involving these functions are passed to the DB2 database, and Unicode string data is manipulated in the database, the results can be unexpected, or an error might occur.

This error might also occur if, for example, an expression of this type is passed directly to the database by using the PASSTHRU function. In this situation, you could modify each expression yourself, as necessary, for the target database.

It is not possible to systematically modify expressions to avoid this problem and IBM App Connect Enterprise does not attempt to do so.

If the Unicode strings do not use any characters that, in UTF8 representation, occupy more than 1 byte each, the functions perform correctly.

Configuring properties to connect to external resources

You can use policies to prepare the environment for external resources, or change connection details for external resources.

About this task

The following topics describe how to prepare your environment to connect to external resources.

Configuring for Enterprise Information Systems

You can use policies to enable the WebSphere Adapters nodes in the integration node runtime environment to connect with Enterprise Information Systems such as SAP, Siebel, and PeopleSoft.

About this task

The following topics describe how to prepare the environment to connect to Enterprise Information Systems, and how to change the connection details for adapters without the need to redeploy them.

Preparing the environment for WebSphere Adapters nodes

Before you can use the WebSphere Adapters nodes, you must set up the IBM App Connect Enterprise runtime environment so that you can access the Enterprise Information System (EIS).

Before you begin

Read "WebSphere Adapters nodes" on page 1050.

About this task

To enable the WebSphere Adapters nodes in the IBM App Connect Enterprise runtime environment, configure the integration server with the location of the EIS provider JAR files and native libraries. (On Windows, the location of the JAR files cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

The integration server configuration file (`server.conf.yaml`) contains a section for connector providers, which defines the location of the required JAR files and native libraries, as you can see in the following example:

```
ConnectorProviders:
  SAPConnectorProvider:
    jarsURL: default_Path     # Set to the absolute path containing the SAP JCo JARs
    nativeLibs: default_Path  # Set to the absolute path containing the SAP JCo libraries
```

By default, the location is set to `default_Path`, which indicates that the JAR files and libraries can be found in the integration server's shared-classes directory.

Procedure

To configure the integration server with the location of required JAR files and libraries, complete the following steps.

1. The WebSphere Adapters nodes require libraries from the EIS vendors. For more information on how to obtain and use these libraries, see "Developing message flows that use WebSphere Adapters" on page 1106.
2. Use a YAML editor to open the `server.conf.yaml` file, then find the `ConnectorProviders` section.

   For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
3. Set the `jarsURL` property to the absolute path where the JAR files are stored.
4. Set the `nativeLibs` property to the absolute path where the native libraries are stored.
5. Save the `server.conf.yaml` file.

What to do next

When you have set up the environment for the WebSphere Adapters nodes, you can perform the preparatory tasks that are listed in "Developing message flows that use WebSphere Adapters" on page 1106.

Changing connection details for SAP adapters

SAP nodes can get SAP connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To

pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the SAP Connection policy to change connection details for an SAP adapter. The SAP node reads all connection properties from the adapter component that it is configured to use. If a SAP Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the SAP policy are described in SAP Connection policy (SAPConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **SAP Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the SAPInput or SAPRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see SAP Connection policy (SAPConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Changing connection details for Siebel adapters

Siebel nodes can get Siebel connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the Siebel Connection policy to change connection details for a Siebel adapter. The Siebel node reads all connection properties from the adapter component that it is configured to use. If a Siebel Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the Siebel policy are described in Siebel Connection policy (SiebelConnection).

You can also connect to different versions of Siebel by creating an EIS Siebel Providers policy and setting the location of the appropriate library files (see "Connecting to different versions of Siebel" on page 1125).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **Siebel Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the SiebelInput or SiebelRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see Siebel Connection policy (SiebelConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Changing connection details for PeopleSoft adapters

PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the PeopleSoft Connection policy to change connection details for a PeopleSoft adapter. The PeopleSoft node reads all connection properties from the adapter component that it is configured to use. If a PeopleSoft Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the PeopleSoft policy are described in PeopleSoft Connection policy (PeopleSoftConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **PeopleSoft Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the PeopleSoftInput or PeopleSoftRequest node (without the `.inadapter` or `.outadapter` suffix).

3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see PeopleSoft Connection policy (PeopleSoftConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Changing connection details for JD Edwards adapters

JDEdwards nodes can get JD Edwards EnterpriseOne connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the JDEdwards Connection policy to change connection details for a JD Edwards adapter. The JD Edwards node reads all connection properties from the adapter component that it is configured to use. If a JDEdwards Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the JD Edwards policy are described in JDEdwards Connection policy (JDEdwardsConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **JD Edwards Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the JDEdwardsInput or JDEdwardsRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see JDEdwards Connection policy (JDEdwardsConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Configuring EIS connections to expire after a specified time

You can configure connections to SAP, Siebel, and PeopleSoft to expire after a specified time by using a policy.

About this task

You can use the `Connection idle timeout` property on a policy to control the number of connections to SAP, Siebel and PeopleSoft by closing connections that have not been used for a specified time.

Procedure

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select an appropriate policy type, such as **SAP Connection** (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the message flow node that you want to override.

   Specify the name of the policy project and the policy on the message flow node in the format {*policyProjectName*}:*PolicyName*.
3. Set the `Connection idle timeout` property on the policy to the number of seconds for which the connection can be idle before it is closed.

   The default value for this property is 0 (zero) seconds, indicating that no timeout occurs. Note that new connections to SAP are opened with different user IDs; therefore do not set this property to zero if you are using identity propagation.
4. Deploy this policy before you start the associated message flow.

Advanced configuration properties when using IBM Sterling Connect:Direct nodes

CDInput and CDOutput nodes can get connection details and staging directories in conjunction with a policy.

Before you begin

- Read "IBM Sterling Connect:Direct overview and concepts" on page 953 for background information.

About this task

Use the Connect:Direct Server policy to change connection details for an IBM Sterling Connect:Direct node (see Connect:Direct Server policy (CDServer)).

Use the following scenarios as examples of how you use the staging directories.

Procedure

- **What should I do if I have IBM App Connect Enterprise and IBM Sterling Connect:Direct on the same machine, and want the files to be staged for output within the IBM App Connect Enterprise work path?**

  You need not set any of the properties; the default settings allow this to happen.
- **Output**
  a) **What should I do if I have IBM App Connect Enterprise and IBM Sterling Connect:Direct on the same machine but want to use a specified location to create files for output?**

  Set the `Integration node path to staging directory` property on the policy to the location that you want. You do not need to set the `Connect:Direct path to staging`

`directory` property, because IBM App Connect Enterprise assumes this to be the same as the `Integration node path to staging directory`.

b) **Can I have IBM App Connect Enterprise and IBM Sterling Connect:Direct on separate machines?**

You can, but you must ensure that both machines have:

– Access to the same shared file system.

– The same timezone setting.

As IBM Sterling Connect:Direct does not support NFS file systems on z/OS, this configuration does not work on z/OS

c) **Can I have this shared file system mounted at different places on my IBM App Connect Enterprise and IBM Sterling Connect:Direct machines for platforms other than Windows?**

Set the `Integration node path to staging directory` property to be the location where it is mounted on the IBM App Connect Enterprise machine, and the `Connect:Direct path to staging directory` property to where it is mounted on the IBM Sterling Connect:Direct machine.

d) **Can I have this shared file system mounted at different places on my IBM App Connect Enterprise and IBM Sterling Connect:Direct machines for Windows platforms?**

You must use the `\\hostname\directory` path syntax to the shared drive, rather than a mapped drive letter. In addition, the user ID that is accessing the `\\hostname\directory` path (the IBM App Connect Enterprise or IBM Sterling Connect:Direct user ID, depending on which one is running on Windows), must have full (write) access to the file system, using the same password.

Set the `Integration node path to staging directory` to be the location where it is mounted on the IBM App Connect Enterprise machine, and the `Connect:Direct path to staging directory` to where it is mounted on the IBM Sterling Connect:Direct machine.

- **Input**

a) **Can I process files in the CDInput node if the IBM App Connect Enterprise and receiving Connect:Direct server are on different machines?**

You can, but you must ensure that both machines have access to the same shared file system and the same time zone setting.

As IBM Sterling Connect:Direct does not support NFS file systems on z/OS, this configuration does not work on z/OS

b) **Can I have this shared file system mounted at different places on my IBM App Connect Enterprise and IBM Sterling Connect:Direct machines for platforms other than Windows?**

Set the `Integration node path to input directory` property to be the location where it is mounted on the IBM App Connect Enterprise machine, and the `Connect:Direct path to input directory` property to where it is mounted on the IBM Sterling Connect:Direct machine.

c) **Can I have this shared file system mounted at different places on my IBM App Connect Enterprise and IBM Sterling Connect:Direct machines for Windows platforms?**

You must use the UNC path syntax to the shared drive, rather than a mapped drive letter. In addition, the user ID that is accessing the UNC path (the IBM App Connect Enterprise or IBM Sterling Connect:Direct user ID, depending on which one is running on Windows), must have full (write) access to the file system, using the same password.

Set the `Integration node path to input directory` property to be the location where it is mounted on the IBM App Connect Enterprise machine, and the `Connect:Direct path to input directory` property to where it is mounted on the IBM Sterling Connect:Direct machine.

Configuring for IMS

You can enable the IMS nodes in the integration node runtime environment to connect with the IMS system.

About this task

The following topics describe how to prepare the environment to connect to the IMS system, and how to change the connection details without the need to redeploy your message flow.

Preparing the environment for IMS nodes

Before you can use the IMS nodes, you must set up the runtime environment so that you can access the IMS system.

Before you begin

Read "IBM Information Management System (IMS)" on page 1146.

About this task

Complete the following steps to ensure that IBM App Connect Enterprise can connect to the IMS system.

Procedure

1. Ensure that IMS Connect is installed and started on the IMS system.
2. Use the **mqsisetdbparms** command to set security details in the integration server store.

   For example, to associate a user ID and password pair with an IMS Connect connection, run the **mqsisetdbparms** command as shown:

```
mqsisetdbparms -w c:\workdir\ACEServ1 -n ims::mySecurityIdentity -u myuserid -p mypassword
```

Preparing the environment for IBM MQ File Transfer Edition nodes

Prepare the file system and queue managers, and determine the name of the integration node agent.

About this task

Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as your integration node if you want to use the capabilities provided by the FTEInput and FTEOutput nodes. For information about installing and using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

- "Preparing the file system" on page 215
- "Preparing the queue manager" on page 216
- "Setting the coordination queue manager" on page 216
- "Naming integration servers" on page 217
- "Determining the agent name" on page 218

Preparing the file system

IBM App Connect Enterprise uses a location in its work path to store transfers to remote agents. It uses another location as the default directory for received files. The high-level directory path for both locations is:

- *workpath*/common/FTE

Ensure that enough space is available here for files that you will transfer to and from the integration node by IBM MQ File Transfer Edition.

Preparing the queue manager

Storage queues that hold file transfer information are owned by the queue manager that is specified on the integration node, and you specify this queue manager by using the **-q** property of the **mqsicreatebroker** command; see “Creating an integration node” on page 114 and command.

You must also create the system queues required by the FTEInput and FTEOutput nodes; see “Creating the default system queues on an IBM MQ queue manager” on page 99. A set of IBM MQ script (MQSC) commands is also provided, to enable you to create artifacts that are required by the agent and coordination manager; for more information, see “Scripts to create artifacts required for IBM MQ File Transfer Edition” on page 218.

Setting the coordination queue manager

When a message flow that contains a IBM MQ File Transfer Edition node is deployed to an integration server, an agent is automatically created and started in that integration server. By default, the agent uses the queue manager that is specified on the integration node as the coordination queue manager.

- If the queue manager specified on the integration node is being used as the coordination queue manager, the integration node configures it as a coordination queue manager.
- If you are using a different queue manager as the coordination queue manager, refer to the WebSphere File Transfer Edition product documentation for information about how to configure it as a coordination queue manager.

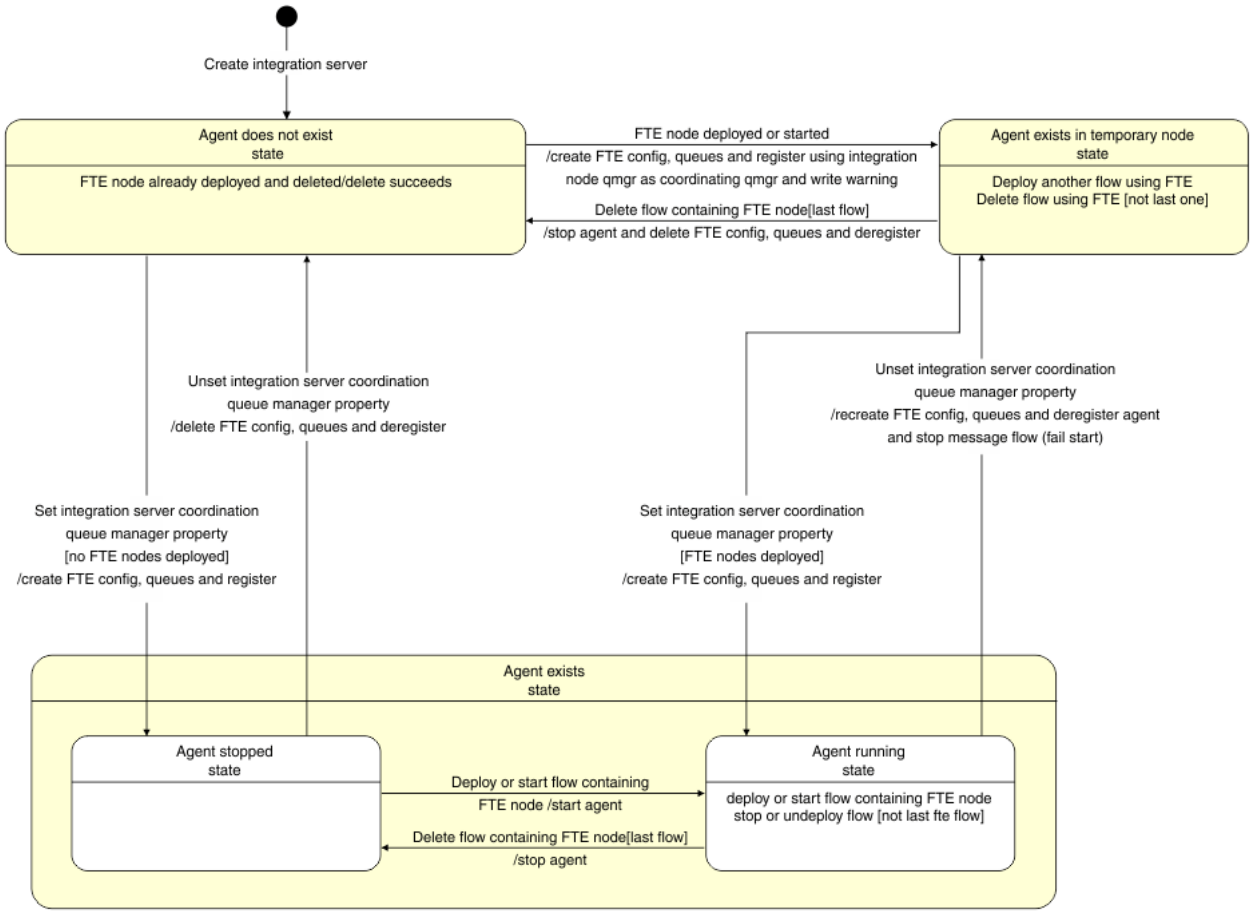
Unless you have previously defined the coordination queue manager, the agent is temporary; it is deleted when the flow is undeployed or the integration node is stopped. This behavior is acceptable in a test environment. However, for production, the administrator must specify the coordination queue manager for the integration server. Specifying a coordination queue manager has the following effects:

Ensures that the correct queue manager is used when the agent is created.

Makes the agent permanent. If a coordination queue manager has been defined, the agent is deleted only after you undefine the coordination queue manager (for example, by setting it to an empty string), and restart the integration server.

A warning is written to the log if the coordination queue manager is not changed from the default.

The following state diagram illustrates how the presence of nodes and a defined coordination queue manager affect the state of the agent.



About this task

Use one of the following methods to set the coordination queue manager.

Naming integration servers

About this task

The integration server name is used to form the queue name for IBM MQ File Transfer Edition queues. Consequently, the names of your integration servers must conform to the rules for naming IBM MQ objects. You cannot deploy a flow that contains a IBM MQ File Transfer Edition node unless this requirement is met. Permitted characters are:

- Uppercase A-Z
- Lowercase a-z (but there are restrictions on the use of lowercase letters for z/OS console support)

  On systems using EBCDIC Katakana you cannot use lowercase characters.
- Numerics 0-9
- Period (.)
- Forward slash (/)
- Underscore (_)
- Percent sign (%)

See the IBM MQ product documentation online for full details of naming requirements.

Determining the agent name

About this task

To send a file to a given integration server, users need to know the name of the agent that the integration node creates. The agent name is derived from *IntegrationNode*.*IntegrationServer*, and is not configurable. The total name length is limited to 28 characters, with a maximum 12 characters for the integration node name, and 15 characters for the integration server. Integration node and integration server names longer than these limits are truncated to form the agent name. The name must be a valid format for generating MQ Series queue name. Ensure that:

- The integration node name is 12 characters or fewer (or at least unique in the first 12 characters).
- The integration server names are 15 characters or fewer (or at least unique in the first 15 characters).
- The integration node and integration servers do not contain any characters that are invalid for queue names.
- The *integrationnode*.*integrationserver* tuples are all unique, even if case is ignored.

The value used is written to the event log in message BIP3358. Use one of the following methods to determine the agent name.

Procedure

Optional: Use the **mqsireportproperties** command.

For example, to display the FTE agent name for integration server FTESAMPLE in integration node INODE:

```
mqsireportproperties INODE -e FTESAMPLE -o FTEAgent -n agentName
```

If the agent has been created, the command returns the agent name. If the agent has not been created, the command returns an empty string.

Scripts to create artifacts required for IBM MQ File Transfer Edition

IBM MQ script (MQSC) commands for use if IBM App Connect Enterprise cannot create all the required artifacts, or you want to create them yourself.

Queues for the FTE agent

Before running the following command, replace *Agent name* with the name of your FTE agent; see "Determining the agent name" on page 218.

```
DEFINE QLOCAL(SYSTEM.FTE.COMMAND.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.DATA.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
```

```
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.REPLY.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.STATE.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.EVENT.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(5000) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHAGT1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
```

```
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHTRN1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHOPS1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHSCH1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHMON1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.AUTHADM1.Agent name) +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(0) +
```

```
 MAXMSGL(0) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
```

Queues for the coordination queue manager

```
DEFINE TOPIC('SYSTEM.FTE') TOPICSTR('SYSTEM.FTE') REPLACE
ALTER TOPIC('SYSTEM.FTE') NPMSGDLV(ALLAVAIL) PMSGDLV(ALLAVAIL)
DEFINE QLOCAL(SYSTEM.FTE) LIKE(SYSTEM.BROKER.DEFAULT.STREAM) REPLACE
ALTER QLOCAL(SYSTEM.FTE) DESCR('Stream for WMQFTE Pub/Sub interface')
* Altering namelist: SYSTEM.QPUBSUB.QUEUE.NAMELIST
* Value prior to alteration:
DISPLAY NAMELIST(SYSTEM.QPUBSUB.QUEUE.NAMELIST)
ALTER NAMELIST(SYSTEM.QPUBSUB.QUEUE.NAMELIST) +
 NAMES(SYSTEM.BROKER.DEFAULT.STREAM+
 ,SYSTEM.BROKER.ADMIN.STREAM,SYSTEM.FTE)
* Altering PSMODE.  Value prior to alteration:
DISPLAY QMGR PSMODE
ALTER QMGR PSMODE(ENABLED)
DEFINE QLOCAL(SYSTEM.FTE.DATABASELOGGER.REJECT) +
 DESCR('Messages rejected by the FTE database logger.') +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(999999999) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(999999999) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
DEFINE QLOCAL(SYSTEM.FTE.DATABASELOGGER.COMMAND) +
 DESCR('Command messages to control the FTE database logger.') +
 DEFPRTY(0) +
 DEFSOPT(SHARED) +
 GET(ENABLED) +
 MAXDEPTH(999999999) +
 MAXMSGL(4194304) +
 MSGDLVSQ(PRIORITY) +
 PUT(ENABLED) +
 RETINTVL(5000) +
 SHARE +
 NOTRIGGER +
 USAGE(NORMAL) +
 REPLACE
```

Preparing the environment for callable flows

To split message flow processing between different locations, you must set up secure data routing.

Before you begin

Read the concept information in “Callable message flows” on page 491.

About this task

You can split message flows so that you do processing in different locations, such as IBM App Connect Enterprise and IBM App Connect on IBM Cloud, which can improve performance and promote reuse.

If you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, your flows communicate by using a Switch server and connectivity agents. The Switch server is a special kind of integration server that routes data. The connectivity agents contain the certificates that your flows require to communicate securely with the Switch server. The connectivity agents must be running in both the integration servers that are running on the integration node where you have deployed your flow.

The steps that you need to take to prepare your environment differ depending on whether you are splitting processing between different integration servers, or between IBM App Connect Enterprise and IBM Integration Bus on Cloud or IBM App Connect on IBM Cloud. For more information, see the following tasks.

Procedure

To prepare the environment for callable flows, complete one of the following tasks.

- “Preparing the environment to split processing between different integration servers” on page 222
- “Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud” on page 225

Preparing the environment to split processing between different integration servers

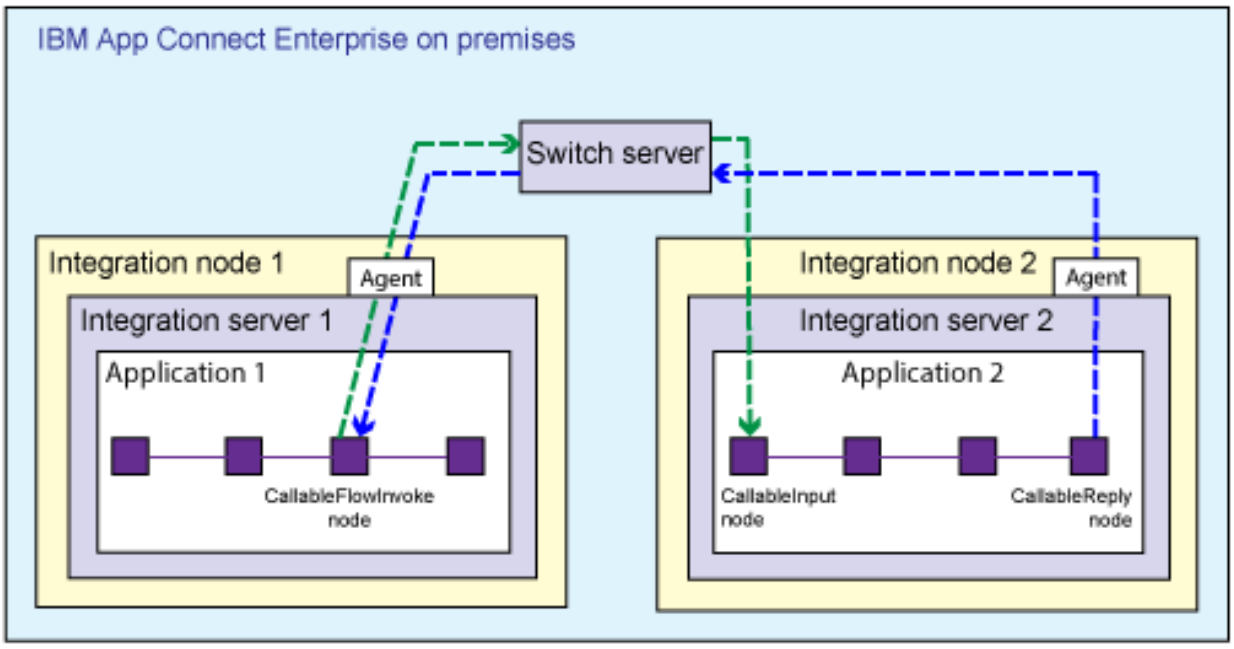
When you split message flow processing between different integration servers, you must configure a data router and a connectivity agent. The integration servers can be running on different integration nodes.

Before you begin

Read the concept information in “Callable message flows” on page 491.

About this task

When you split processing between different integration servers, your flows communicate securely by using a Switch server and connectivity agents. The Switch server is a special kind of integration server that routes data. The connectivity agents contain the certificates that your flows require to communicate securely with the Switch server. A connectivity agent must be running in each integration server where you have deployed message flows.



If your callable flows and the flows that call them are ***all*** deployed on premises, you must create and configure the Switch server on premises. You must run a command that creates some configuration files. You use one of the configuration files to create the Switch server, and the other file to configure connectivity agents for each integration server where you have deployed callable flows or flows that call them. The steps for this procedure are given in this topic.

This on-premises scenario is supported on Windows and Linux only.

If you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, the Switch server is created and managed for you in the cloud. You must download an agent configuration file from the cloud, and use it to configure the on-premises connectivity agent for each integration server. For more information about configuring connectivity agents for use with IBM App Connect on IBM Cloud, see "Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud" on page 225.

Procedure

To prepare the environment to split processing between integration servers on premises, complete the following steps.

1. Start an IBM App Connect Enterprise command environment.
2. Create the required configuration files, by running the **iibcreateswitchcfg** command.
   You need to run the command in a directory for which you have permission to write. Alternatively, you can specify an output directory. For example, the following command creates the configuration files and saves them in the `temp` directory.

   Windows On Windows:

   ```
   iibcreateswitchcfg /output c:\temp
   ```

   Linux On Linux:

   ```
   iibcreateswitchcfg -output /temp
   ```

   If this command is successful, you see the following responses:

   ```
   Generated self signed certificate file 'c:\temp\adminClient.p12'
   Generated switch configuration file 'c:\temp\switch.json'
   ```

```
Generated agentx configuration file 'c:\temp\agentx.json'
```

This command creates two JSON configuration files, and a certificate, which is reserved for future use. One file (`switch.json`) is used to create the Switch server. The other file (`agentx.json`) is used to configure the agent for each integration server where your flows are deployed.

3. Create the Switch server, by running the **iibswitch** command to use the configuration file (`switch.json`) that you created in step 2.

   Windows For example, on Windows:

   ```
   iibswitch create switch /config c:\temp\switch.json
   ```

   Linux On Linux:

   ```
   iibswitch create switch -c /temp/switch.json
   ```

   If the command is successful, you see a "Successful command completion" response. The Switch server is created in a special integration node called ACESWITCH_NODE_V11.

4. Optional: To test that the Switch server is created and running, run the following command: **mqsilist** ACESWITCH_NODE_V11.

   If the Switch server is running, you see the following response:

   ```
   BIP1286I: Integration server 'IIBSWITCH_SERVER' on integration node 'ACESWITCH_NODE_V11' is
    running.
   ```

5. Configure a connectivity agent for each integration server where you have deployed callable flows or flows that call them.

   You can configure the agent by copying the configuration file, `agentx.json` (that you created in step 2), to the server's `iibswitch/agentx` directory.

   - For an independent integration server: *work directory*`/config/iibswitch/agentx`
   - For an integration server under an integration node: *$MQSI_WORKPATH*`/config/`*Node_name*`/`*Server_name*`/iibswitch/agentx`

   For each affected integration server, the agent should automatically establish a connection to the Switch server. For example, in the App Connect Enterprise console for an independent integration server you should see the following messages:

   ```
   The connection agent for remote callable flows has established a connection to the Switch
    server with URL 'wss://localhost:9011'.
   The integration server component 'agentx' has been started.
   ```

   Similarly, in the Windows Application log, you should see the following messages for an integration server (N101ISAA) under an integration node (NODEIPL01):

   ```
   ( NODEIPL01.N101ISAA ) The connection agent for remote callable flows has established a
    connection to the Switch server with URL ''wss://localhost:9011''.

   A connection agent for remote callable flows has been configured on this integration server.
    The connection agent has successfully connected to the Switch server.

   No response required.
   ----------------------------------------------------------------------------------------------
   ( NODEIPL01.N101ISAA ) The integration server component ''agentx'' has been started.

   The specified component has been started and is available for use.

   No response required.
   ```

Results

A message flow that is deployed to one configured integration server can now communicate securely with a message flow on another configured integration server. For more information about developing these message flows, see “Developing synchronously callable message flows” on page 600.

Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud

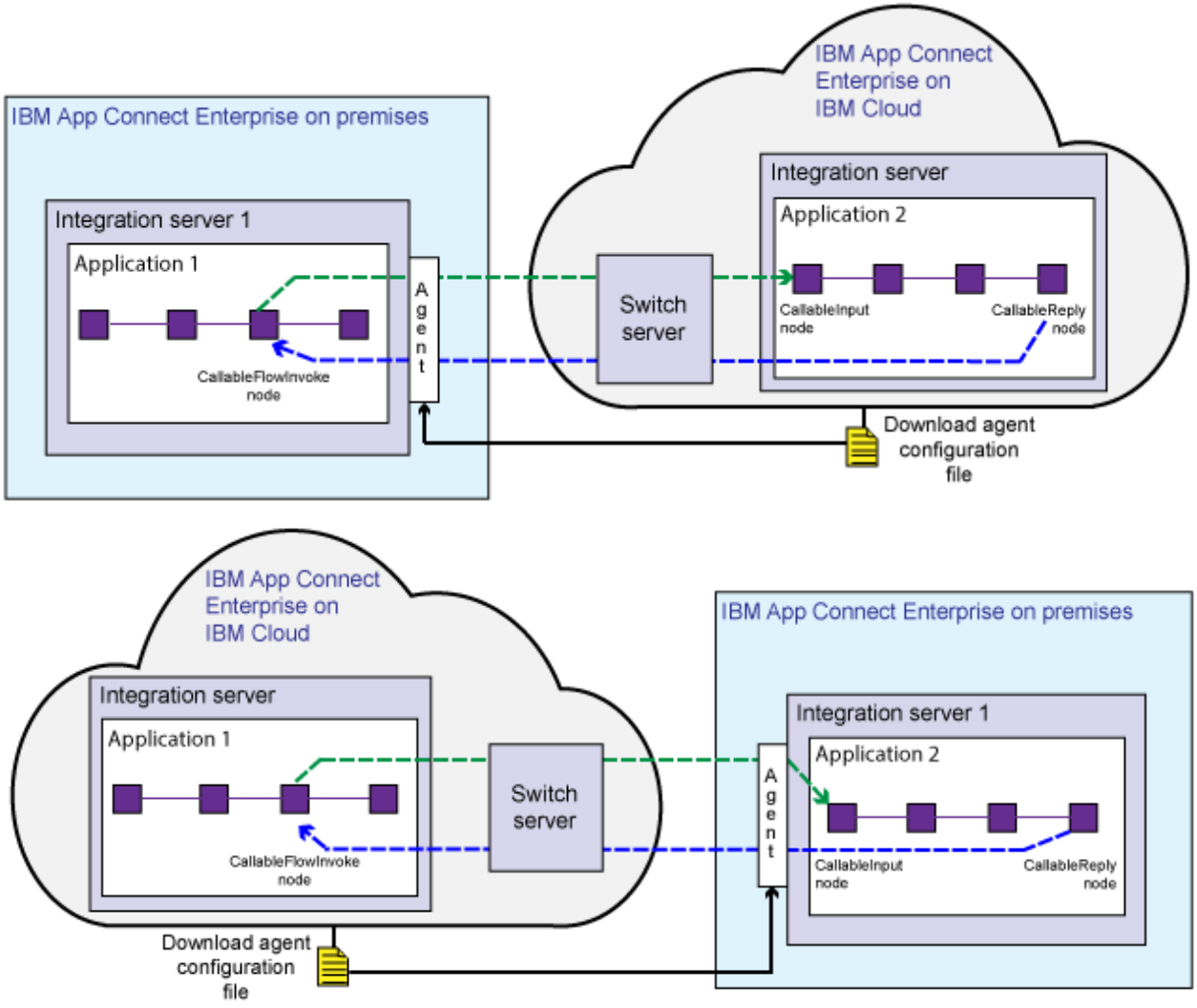
When you split message flow processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, you must configure a connectivity agent.

Before you begin

These instructions assume that you have IBM App Connect Enterprise installed on premises, and that you have created an App Connect service in IBM Cloud.

About this task

When you split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, your flows communicate by using a Switch server and connectivity agent. The Switch server, which routes data, is managed for you by IBM App Connect on IBM Cloud. The connectivity agent contains the certificates that your flows require to communicate securely with the Switch server. A connectivity agent must be running in the IBM App Connect Enterprise integration server where you have deployed your on-premises message flows.



The Switch server is already created in IBM App Connect on IBM Cloud. You must download an agent configuration file from the cloud, and use it to configure the on-premises connectivity agent. Your callable flows use this agent to communicate with each other securely, through the Switch server.

Procedure

To prepare the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, complete the following steps.

1. In IBM App Connect on IBM Cloud, click **Callable flows**  .

   If you have not configured callable flows before, the list will be empty.
2. Click **Connect callable flows**.

   A dialog box opens to set up your agent.
3. Click **Download the configuration** and save the agent configuration file (`agentx.json`) to convenient location, and then use it to configure the on-premises integration servers.

   Copying the `agentx.json` file, to the on-premises server's `iibswitch/agentx` directory.

   - For an independent integration server: *work directory*`/config/iibswitch/agentx`
   - For an integration server under an integration node: *$MQSI_WORKPATH*`/config/`*Node_name*`/`*Server_name*`/iibswitch/agentx`

   For each affected integration server, the agent should automatically establish a connection to the Switch server. For example, in the App Connect Enterprise console for an independent integration server you should see the following messages:

   ```
   The connection agent for remote callable flows has established a connection to the Switch
    server with URL 'wss://ibm-sw-lfcxnlolwumbamsnq.eu-gb.ace.ibm.com:443/'.
   The integration server component 'agentx' has been started.
   ```

   Similarly, in the Windows Application log, you should see the following messages for an integration server (N101ISAA) under an integration node (NODEIPL01):

   ```
   ( NODEIPL01.N101ISAA ) The connection agent for remote callable flows has established
    a connection to the Switch server with URL ''wss://ibm-sw-lfcxnlolwumbamsnq.eu-
   gb.ace.ibm.com:443/''.

   A connection agent for remote callable flows has been configured on this integration server.
    The connection agent has successfully connected to the Switch server.

   No response required.
   ----------------------------------------------------------------------------------------------
   ( NODEIPL01.N101ISAA ) The integration server component ''agentx'' has been started.

   The specified component has been started and is available for use.

   No response required.
   ```

   Note: If a message flow on one on-premises integration server calls a message flow on other on-premises integration servers, and one of those integration servers also uses the callable flows technique to interact with flows in IBM App Connect on IBM Cloud, you need to copy the downloaded `agentx.json` file to all those on-premises integration servers (so that they all continue to use the same Switch server, now on cloud). For example, consider the initial on-premises scenario where a flow on server A calls a flow on server B, with both servers configured with the local `agentx.json` (to use the on-premises Switch server). Later, a flow in IBM App Connect on IBM Cloud is added to call the flow on server B, so server B needs to be configured with the `agentx.json` downloaded from the cloud (to use the on-cloud Switch server). Also, to continue to use the same Switch server as server B, server A needs to be configured with the `agentx.json` downloaded from the cloud (to use the on-cloud Switch server)

4. In IBM App Connect on IBM Cloud, you can check that the on-premises agent can connect to the Switch server in the cloud.

   You can perform the following checks:

   - Refresh the Callable flows page. This should list all the callable and calling flows on premises and on cloud; for example:
   - In the **Connect callable flows** dialog, click **Test your agent**. This should return the number of on-premises agents found.

   If the test is successful, close the dialog box.

Results

The callable flows technique can be used between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, and between integration servers in IBM App Connect Enterprise configured to use the on-cloud Switch server.

For more information about developing these message flows, see "Developing synchronously callable message flows" on page 600.

Configuring for JMS

You can use policies to configure IBM App Connect Enterprise for JMS.

About this task

The following topics describe how to configure IBM App Connect Enterprise for JMS.

Enabling a JMS provider's proprietary API

Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

About this task

For example, BEA WebLogic uses a component called a *Client Interposed Transaction Manager* to allow a JMS client to obtain a reference to the XAResource that is associated with a user transaction.

If the WebSphere IBM Integration JMS nodes use BEA WebLogic as the JMS provider, and the nodes must participate in a globally coordinated message flow, you must modify the policy properties that are associated with that vendor (see JMS Providers policy (JMSProviders)). The following table shows the properties that have been added to the policy for BEA WebLogic.

JMS provider	Property	Purpose	Default value
BEA_WebLogic	`Proprietary API handler`	The name of the IBM supplied Java class to interface with a JMS provider's proprietary API.	`com.ibm.broker.apihandler.BEAWebLogicAPIHandler`
	`Proprietary API attribute 1`	The Initial Context Factory class name for the vendor	`weblogic.jndi.WLInitialContextFactory`
	`Proprietary API attribute 2`	The URL of the WebLogic bindings	*URL JNDI bindings*
	`Proprietary API attribute 3`	The DNS name of the JMS server	*Server name*

In the list of JMS provider policy templates, the name of the IBM supplied Java class is set to the default value for the `Proprietary API handler` property. Typically, you do not need to change this value, unless you are instructed to do so by an IBM Service team representative.

Procedure

- Use the Policy editor to modify values of the properties for this JMS provider.
- If you have defined a JMS provider policy, set the value for the `Proprietary API handler` property manually.

Connecting to different versions of the same JMS provider

To use different versions of the same JMS provider, create a policy for the JMS provider, and set the location of the JAR files to a unique path.

Procedure

To connect to two versions of the same JMS provider (for example, JBoss), complete the following steps.

1. Create a separate JMS Providers policy for each version of the JMS provider.

   For more information, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.
2. Set the **`Type 4 driver class JARs URL`** property of the JMS Providers policy to a unique path.

   For a description of the properties of the JMS Providers policy, see JMS Providers policy (JMSProviders).

Configuring the JMSInput node for batch message processing

Configure JMS message flows to send a batch acknowledgment for receipt of non-transactional JMS messages.

About this task

When the JMSInput node works in non-transactional mode, receipt and acknowledgment of messages take place in one step, followed by the message processing. In some scenarios, this acknowledgment response to the JMS server for each message might create an unacceptable level of network traffic. For example, using this messaging model to receive JMS messages across a wide area network that is already handling large volumes of traffic might result in non-optimal throughput rates for JMS messages.

The JMSInput node can acknowledge message receipt in batches rather than individually for non-transactional messages. Batch acknowledgment is enabled by using the JMS Providers policy properties **`clientAckBatchSize`** and **`clientAckBatchTime`**. You can set these properties separately, or use them together, to tune the number of messages that are received and processed by the node before an acknowledgment response is returned to the source JMS server.

clientAckBatchSize
: This an integer value that represents the threshold number of messages received before the batch acknowledgment is sent.

clientAckBatchTime
: This is an integer value that represents the length, in milliseconds, of a repeating interval. At the end of each interval a batch acknowledgment is sent for all unacknowledged non-transactional JMS messages that were received during the preceding interval.

A batch acknowledgment is also sent when:

- There are no more input messages on the JMS server
- An error occurs during message processing. In this case, all previous messages in the batch that were successfully processed are first acknowledged, before handling the error.
- The message flow stops.

To disable batch acknowledgment, set both **`clientAckBatchSize`** and **`clientAckBatchTime`** to `0`.

Configuring for email

You can use policies to enable the EmailOutput node and EmailInput node in the IBM App Connect Enterprise runtime environment to connect with email servers.

About this task

The following topics describe how to prepare the environment to connect to email servers, and how to change the connection details without the need to redeploy your message flow.

- "Changing connection information for the EmailOutput node" on page 946
- "Changing connection information for the EmailInput node" on page 951

Configuring properties for TCP/IP

If you are deploying message flows that contain TCP/IP nodes, and those nodes use a policy, create and deploy your TCPIPClient or Server policies before the message flows are deployed.

Before you begin

For background information about the TCP/IP transport, see "TCP/IP data transfer" on page 911 and "Connection management" on page 917.

Procedure

- Create a TCPIPClient or TCPIPServer policy by using the Policy editor in the IBM App Connect Enterprise Toolkit, as described in "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.

  For more information about these policies, see TCPIP Client policy (TCPIPClient) and TCPIP Server policy (TCPIPServer).

Configuring internal resources required by message flows

You can use policies to configure the internal resources that are required by some message flows.

About this task

The following topics describe how to configure the storage of events for certain message flow nodes.

Configuring the storage of events for aggregation nodes

You can use an Aggregation policy to control the storage of events for AggregateControl and AggregateReply nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using aggregation on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the aggregation nodes. If you are using aggregation on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing aggregation nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry time of an aggregation:

Procedure

1. Create the storage queues to be used by the aggregation nodes.

   The following queues are required:

   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET.1 are valid queue prefixes, but .SET1 and SET1. are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Create an Aggregation policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific aggregation or with all aggregations in an integration server. If the policy is to be used with a specific aggregation, create the policy with the same name as the name that you specify in the `Aggregate name` property on the AggregateControl and AggregateReply nodes.

      To specify a default Aggregation policy for all message flows that are deployed to an integration server, set the **Aggregation** property in the `server.conf.yaml` file to the name of an Aggregation policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy

project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format {*policyProjectName*}:*PolicyName*

b) Set the **Queue prefix** property of the Aggregation policy to the required value (see Aggregation policy).

c) Optional: Set the **Timeout** property of the Aggregation policy to control the expiry time of an aggregation.

If you delete the Aggregation policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.

3. In the AggregateControl and AggregateReply nodes, ensure that the name of the Aggregation policy is the same as the name that is specified in the `Aggregate name` property on the **Basic** tab; for example, `myAggregation`.

   If there is no Aggregation policy with the same name as the `Aggregate name` node property value, and if there is a default Aggregation policy specified in the `server.conf.yaml` file, that Aggregation policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Configuring the storage of events for Collector nodes

You can use a Collector policy to control the storage of events for Collector nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using message collections on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the Collector node. If you are using message collections on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow that contains Collector nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry for the collection:

Procedure

1. Create the storage queues to be used by the Collector node.

   The following queues are required:

   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET.1 are valid queue prefixes, but .SET1 and SET1. are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Create a Collector policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific collection or with all collections in an integration server. If you are creating a policy to be used with a specific collection, ensure that the name of the policy is the same as the name that you specify in the `Policy` property on the Collector node.

      To specify a default Collector policy for all message flows that are deployed to an integration server, set the **`Collector`** property in the `server.conf.yaml` file to the name of a Collector policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*
   b) Set the **`Queue prefix`** property of the Collector policy to the required value.
   c) Optional: Set the **`Collection expiry`** property of the Collector policy to an appropriate value (for example, 60).

   If you delete a Collector policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.
3. In the Collector node, if the policy is to be used for a specific collection, specify the name of the policy in the `Policy` property on the **Advanced** tab; for example, `myCollectorService`.

   If you do not set the `Policy` property, and if there is a default Collector policy specified in the `server.conf.yaml` file, that policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Configuring the storage of events for Resequence nodes

You can use a Resequence policy to control the storage of events for Resequence nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using Resequence nodes on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the Resequence node. If you are using Resequence nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent

integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing Resequence nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Collector node.

However, you can control the queues that are used by different Resequence nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Resequence policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the timeout and the start and end of the sequence:

Procedure

1. Create the storage queues to be used by the Resequence node.

   The following queues are required:

   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET`.`1 are valid queue prefixes, but `.`SET1 and SET1`.` are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Create a Resequence policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific sequence or with all sequences in an integration server. If you are creating a policy to be used with a specific sequence, ensure that the name of the policy is the same as the name that you specify in the `Policy` property on the Resequence node.

      To specify a default Resequence policy for all message flows that are deployed to an integration server, set the **Resequence** property in the `server.conf.yaml` file to the name of a Resequence policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*
   b) Set the **Queue prefix** property of the Resequence policy to the required value (see Resequence policy).
   c) Optional: Set the **Missing message timeout**, **Start of sequence**, and **End of sequence** properties of the Resequence policy.

   If you delete the Resequence policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.

3. In the Resequence node, if the policy is to be used for a specific sequence, specify the name of the policy on the **Advanced** tab; for example, `myResequenceService`.

   If you do not set the `Policy` property, and if there is a default Resequence policy specified in the `server.conf.yaml` file, that policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Configuring the storage of events for timeout nodes

You can use a Timer policy to control the storage of events for TimeoutNotification and TimeoutControl nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using TimeoutControl and TimeoutNotification nodes on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by these nodes. If you are using TimeoutControl and TimeoutNotification nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing TimeoutControl or TimeoutNotification nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **`iib_createqueues`** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer policy to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event states:

Procedure

1. Create the storage queue to be used by the timeout nodes.

   The following queue is required:

   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET`.`1 are valid queue prefixes, but `.`SET1 and SET1`.` are invalid.

   If you do not create the storage queue, IBM App Connect Enterprise creates the queue when the node is deployed; this queue is based on the default queue. If the queue cannot be created, the message flow is not deployed.

2. Create a Timer policy (see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327).
   a) You can create a policy to be used with either specific timeout requests or with all timeout requests in an integration server. If the policy is to be used with specific timeout requests, create the policy with the same name as the `Unique identifier` property on the TimeoutNotification and TimeoutControl nodes.

      To specify a default Timer policy for all message flows that are deployed to an integration server, set the **`Timer`** property in the `server.conf.yaml` file to the name of a Timer policy. For information about setting properties in the `server.conf.yaml` file, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*.
   b) Set the **`Queue prefix`** property of the Timer policy to the required value (see Timer policy).

   If you delete the Timer policy, the storage queue is not deleted automatically when the policy is deleted, so you must delete it separately.
3. In the TimeoutNotification and TimeoutControl nodes, ensure that the name of the Timer policy is the same as the name specified in the `Unique Identifier` property on the **Basic** tab; for example, `myTimer`. Specify the name of the policy project and the policy on the message flow node in the format *{policyProjectName}:PolicyName*. If there is no Timer policy with the same name as the `Unique Identifier`, and if there is a default Timer policy specified in the `server.conf.yaml` file, that Timer policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Configuring the XPath cache

The integration server XPath cache size might become a performance bottleneck for customers who use many XPath expressions. Altering the size of the XPath cache might improve message flow performance.

An integration server (or 'execution group' in WebSphere Message Broker Version 8.0 and earlier versions) keeps a cache of compiled XPath expressions to help reduce the processor usage of parsing and re-creating XPath expressions that are used repeatedly during message flow execution. This cache is shared by all message flows within an integration server. The default size of this cache is 10000 elements, and this is expected to be sufficient for most customer configurations.

It might be necessary to alter the size of this cache for optimal message flow performance if hundreds or thousands of XPath expressions are created for each Message Flow invocation. In a highly multi-threaded environment where many XPath expressions are evaluated on each message flow invocation, it might be necessary to disable the cache to remove thread contention on the cache. From IBM Integration Bus Version 10.0 onwards, the XPath cache has its own resource manager **`ComIbmXPathCache`**; run the following command to return the status:

```
mqsireportproperties <INode> -e default -o ComIbmXPathCache -r
```

where *<INode>* is your integration node. The following output is an example of that returned from the command:

```
ComIbmXPathCache
 uuid='ComIbmXPathCache'
 userTraceLevel=''
 traceLevel=''
 userTraceFilter=''
 traceFilter=''
 vrmfIntroducedAt=''
 resourceGroup=''
 mode='enabled'
 maximumSize='10000'
```

```
minimumSize='6000'
currentSize='0'
entryWarningThreshold='1000'
```

The **maximumSize** property controls the maximum number of compiled XPath expressions that can be stored in the XPath cache. When the XPath cache reaches this limit of compiled XPath expressions, entries are evicted from the cache based on the following criteria:

- How often they were used.
- When they were last used.

You can change the value of the **maximumSize** property with the following command:

```
mqsichangeproperties <INode> -e default -o ComIbmXPathCache -n maximumSize -v 20000
```

The **minimumSize** property controls the minimum number of compiled XPath expressions that can be stored in the XPath cache. When the XPath cache reaches this number of compiled XPath expressions, the number of entries does not fall any lower when entries are evicted from the cache. You can change the value of the **minimumSize** property with the following command:

```
mqsichangeproperties <INode> -e default -o ComIbmXPathCache -n minimumSize -v 1000
```

The **currentSize** property reports the number of compiled XPath expressions that are stored in the XPath cache. This property is read-only, and must not be set with the **mqsichangeproperties** command.

The **mode** property controls whether the cache is enabled or disabled. When the cache is disabled, no XPath expressions are cached. This can improve performance in highly multi-threaded environments where each message flow invocation creates unique XPath expressions, which are only used once. To disable the cache, set the mode property to `disabled`. You can change the value of the **mode** property with the following command:

```
mqsichangeproperties <INode> -e default -o ComIbmXPathCache -n mode -v disabled
```

The **entryWarningThreshold** property controls how frequently activity log messages are emitted by the XPath cache as it grows in size. The XPath cache emits an activity log message that gives updates about the size of the cache whenever the size of the cache divided by the value of **entryWarningThreshold** equals 0. The default value of the **entryWarningThreshold** property is 1000 and is ten per cent of the default maximum size of the cache. The cache also emits an activity log message when the cache reaches the maximum size and a cache flush occurs. If the value of the **maximumSize** property is changed, the value of the **entryWarningThreshold** property might also need to be changed. You can change the value of the **entryWarningThreshold** property with the following command:

```
mqsichangeproperties <INode> -e default -o ComIbmXPathCache -n entryWarningThreshold -v 20
```

Configuring monitoring event sources by using a monitoring profile

You can create a monitoring profile to configure your message flows to emit monitoring events.

Before you begin

Read the following topics:

- "Monitoring basics" on page 2772
- "Configuring monitoring for message flows" on page 2788

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event.

About this task

To configure monitoring events by using a monitoring profile, complete the following tasks.

1. “Creating a monitoring profile” on page 2793
2. “Applying and activating a monitoring profile” on page 2798

Changing locales

You can change the locale for the system on which a runtime component is installed.

About this task

The way in which you change the locale depends on the operating system:

- “Changing your locale on Linux and UNIX systems” on page 237
- “Changing your locale on Windows” on page 239

IBM App Connect Enterprise uses code page converters to support character sets from different environments. “Code page converters” on page 239 describes what a code page converter is, and how to generate new converters.

Changing your locale on Linux and UNIX systems

You can change your system locale on UNIX and Linux systems.

About this task

You can set environment variables to control the system locale. You can set these variables to be system-wide, or on a per-session basis:

LC_ALL
Overrides all LC_* environment variables with the given value

LC_CTYPE
Character classification and case conversion

LC_COLLATE
Collation (sort) order

LC_TIME
Date and time formats

LC_NUMERIC
Non-monetary numeric formats

LC_MONETARY
Monetary formats

LC_MESSAGES
Formats of informative and diagnostic messages, and of interactive responses

LC_PAPER
Paper size

LC_NAME
Name formats

LC_ADDRESS
Address formats and location information

LC_TELEPHONE
Telephone number formats

LC_MEASUREMENT
Measurement units (Metric or Other)

LC_IDENTIFICATION
Metadata about the locale information

LANG
The default value, which is used when either LC_ALL is not set, or an applicable value for LC_* is not set

NLSPATH
Delimited list of paths to search for message catalogs

TZ
Time zone

LC_MESSAGES and NLSPATH are the most important variables to the integration node. These variables define the language and location of response messages that the integration node uses. The integration node profile file, `mqsiprofile`, sets NLSPATH. Either you, or your system must set LC_MESSAGES. The value set in LC_MESSAGES must be a value that is installed on your machine and that the integration node recognizes. LC_CTYPE is also important to the integration node because it defines the character conversion that the integration node performs when interacting with the local environment.

Before setting these variables, check that the language and code page are installed on your machine, and are supported by IBM App Connect Enterprise.

You can use the command **`locale`** to show your current locale. The command **`locale -a`** displays all the locales that are currently installed on the machine. Make sure that the locale you select for LANG and LC_ALL is in the list that is returned by the command **`locale -a`**. The values that `locale` uses and returns are case sensitive, therefore copy them exactly when assigning them to an environment variable.

For information on languages and code pages supported by IBM App Connect Enterprise, see Supported code pages.

If you use common desktop environment (CDE), use this environment to set the locale instead of setting LANG and LC_ALL directly. The NLSPATH variable respects either method.

For example, to set IBM App Connect Enterprise to run in a UTF-8 environment set the following values in the profile:

```
LANG=en_US.utf-8
LC_ALL=en_US.utf-8
```

where en_US sets the language, and `utf-8` sets the code page.

When you start an integration node component, the locale of that component is inherited from the shell in which it is started. The integration node component uses the LC_MESSAGES environment variable as the search path in the NLSPATH environment variable (LC_MESSAGES is set when variable LC_ALL is exported).

Messages are sent to the syslog in the code page set by this locale. If you have multiple integration node that write to this syslog, their messages are in the code page of the locale in which they were started, for example:

locale	syslog code page	ccsid
pt_BR	iso8859-1	819
pt_BR	ibm-850	850
pt_BR	utf-8	1208

Set the locale of the user ID that runs the syslog daemon to one that is compatible with the locales of all integration node that write to the syslog on that system, for example, `utf-8`. For compatibility, you can set the default locale. On Solaris, set the LANG and LC_ALL variables in `/etc/default/init`. On AIX and Linux, these variables are in `/etc/environment`. This task is not required on HP-UX.

For full-time zone support in the integration node, set the TZ variable using Continent/City notation. For example set TZ to Europe/London to make London, England the time zone, or set it to America/New_York to make New York, America the time zone.

If you want to add a new locale, refer to the operating system documentation for information about how to complete that task. If the code page of the new locale is not supported by IBM App Connect Enterprise you must add it by "Generating a new code page converter" on page 240.

Changing your locale on Windows

Change your system locale on Windows to view objects and information in a different language or code page.

About this task

Integration node are started as services on Windows, and are therefore influenced by the system locale. The command-line functions are influenced by the locale that is set for the current user. IBM App Connect Enterprise on Windows has all locale information installed by default. However, you might have to install additional locale packages, if prompted to do so by the Windows operating system.

To change locale, use one of the following methods:

- Install a locale-specific operating system.
- Alter the system or user locale by selecting *Regional Settings* in the Control Panel.

Messages are sent to the Event Log in the code page set by the current locale.

You can use the **chcp** command to change the active console code page. Enter the command at a command prompt; if you enter **chcp** without a parameter, it displays the current setting. If you enter it with a code page, it changes the locale to that code page.

For example, to check the current code page setting:

```
C:\>chcp
Active code page: 437
```

The current page is displayed (437 represents US-ASCII). If you want to change the value to GB18030, enter:

```
C:\>chcp 54936
Active code page: 54936
```

Before you use a code page, search for `windows-`*`number`* where *`number`* is the active code page you want to use in the list of Supported code pages. If the code page is not in the list, either use a code page that is in the list, or generate a new code page converter.

Code page converters

Integration nodes complete string operations in Universal Character Set coded in 2 octets (UCS-2). If incoming strings are not encoded in UCS-2, they are converted to UCS-2 on arrival.

The integration node uses international components for Unicode (ICU) code page converters to convert data. The Unicode Consortium has further information on Unicode.

A code page converter is a mapping from the byte sequence in one code page to a serialized representation of UCS-2, known as UCS Transformation Format 16-bit form (UTF-16). A code page converter allows the integration node to create a UCS-2 representation of an incoming string.

When you handle UTF-16 data, CCSIDs 1200, 13488 and 17584 are treated differently to others. Traditionally, in ICU usage, the endian encoding of these CSSIDs was platform-specific, and IBM App Connect Enterprise uses an encoding parameter with these CSSIDs. You can specify the encoding parameter as MQENC_INTEGER_REVERSED to use these CCSIDs to explicitly produce little endian data.

Consider this example of the use of a code page converter. A message comes in on a queue from z/OS, with the IBM MQ CCSID field set to 1047 (LATIN-1 Open Systems without euro). The integration node looks up ibm-1047 and uses the resulting converter to create a UCS-2 representation for internal use.

If you try to convert from a Unicode to a Non-Unicode character set, the following errors might occur:

- The target buffer is too small. This error causes a recoverable exception, which you can handle; alternatively, the message is rolled back.
- A code point in the source does not have an equivalent value in the target. At first, fallback mappings are attempted (for example, if you are converting to Japanese, a backslash (\) can be mapped to a yen (¥) if the conversion supplies it as a fallback mapping). If fallback mappings are not present, a recoverable exception is thrown. You can handle the exception, or the message is rolled back.

  The MRM parser substitutes invalid code points with substitution characters.

IBM App Connect Enterprise currently supports the code pages that are listed in Supported code pages. If you need support for an additional code page, or if you require a different variant of a code page, you can extend the integration node to support this code page.

Generating a new code page converter

Generate a code page converter to handle conversions of data that belongs to a code page that is not in the default set of code pages that are provided by IBM App Connect Enterprise.

Before you begin

- Read "Code page converters" on page 239, which provides information about what a code page converter is, and about the code pages that IBM App Connect Enterprise supports.
- If you apply a fix pack to IBM App Connect Enterprise that increases the ICU version, recompile the code page converters that are used by IBM App Connect Enterprise message flows or the DFDL parser. (The ICU version is listed in the fix pack document on the IBM Support site.)

About this task

This information is split into the following sections:

- "Making a new code page converter" on page 240
- "Making the new code page converter available to IBM App Connect Enterprise message flows" on page 241
- "Making the new code page converter available to the DFDL parser" on page 241

Making a new code page converter

Procedure

1. Create or find a mapping data file with the file extension `.ucm` for the converter that you require.

   You can download `.ucm` files from the ICU Character set mapping files archive. These mapping data files are available and can be modified without restriction.

   An example mapping data file is `ibm-1284_P100-1996.ucm`. (ICU is an external open source project, not an IBM tool.)
2. Rename the `.ucm` to a file name with the format `ibm-`*`number`*`.ucm` where *number* is a number that you choose to identify the code page. Make sure that this number is not already used in one of the Supported code pages.
   For example, you could rename `ibm-1284_P100-1996.ucm` to `ibm-1284.ucm`.
3. Go to ICU downloads, download the source code and build binaries for your system. If you have problems building the converter, see the ICU user guide.
4. Extract the files from the binary distribution archive into a temporary directory.

5. Copy the library and binary files to a directory in the environment PATH and LIBPATH. (Alternatively, copy the library and binary files to directory that is not temporary, and modify the environment PATH and LIBPATH to include this directory.)
6. One of the extracted files is `makeconv.exe`; use this **makeconv** tool to convert the mapping data file (`.ucm` files) into a binary converter file (`.cnv` file), by entering the following command:

   ```
   makeconv mapping_file.ucm
   ```

   where *mapping_file.ucm* is the mapping data file that you are using.

   The name of the binary converter file that **makeconv** produces is:

   ```
   mapping_file.cnv
   ```

   where *mapping_file.cnv* is the name of the mapping data file that was converted.

   To make the `.cnv` file for `ibm-1284.ucm`, use the following command:

   ```
   makeconv ibm-1284.ucm
   ```

Making the new code page converter available to IBM App Connect Enterprise message flows

Procedure

1. Copy the file with the file extension `.cnv` for the code page that you need, into a directory that IBM App Connect Enterprise can access.

   The name and location of the file is of the form

   ```
   ibm-1284.cnv
   ```

   and is located in the *$ICU_DATA/icudt##<platform-suffix>* directory, where *icudt##* is the version of ICU, which you can find in the fix pack document on the IBM Support site; for example, *icudt51* for IBM Integration Bus version 10.0.0.11. *<platform-suffix>* is one of the following values:

   - `l` for little-endian ASCII platforms
   - `b` for big-endian ASCII platforms
   - `e` for EBCDIC platforms
2. Optional: If you do not want the new code page converter to be in the same location as other ICU data, you must associate the broker with the new directory where the converter is stored (the directory added must contain the full path, not including the `icudt48x` subdirectory):
   - To create a new broker that is associated with the converter, include the **-c** parameter on the **mqsicreatebroker** command.
   - To affect all the products and the broker command-line tools that are using ICU, add the *directory* to the **ICU_DATA** environment variable. If you have used the **mqsicreatebroker** command to specify the code page converter to be used, the broker ignores the **ICU_DATA** value.

     Note: To ensure consistent behavior in all components, modify the **ICU_DATA** environment variable.

Making the new code page converter available to the DFDL parser

Before you begin

The DFDL component includes its own copy of the ICU libraries. This copy might be a different version level to those libraries used by IBM App Connect Enterprise. In this case, if the new code page is to be used for DFDL parsing, the new code page converter must also be copied into a specific location that can be accessed by DFDL. The DFDL component uses the environment variable **ICU_DATA** as a root location

to search for convert tables. This environment variable should already have been set for you by running **mqsiprofile**.

Procedure

1. Copy the new code page convert `.cnv` file to a subdirectory beneath the location that is specified in the **ICU_DATA** environment variable. This subdirectory might need to be created, and its name must match the version of the ICU libraries that are supplied with the DFDL component.

   For IBM App Connect Enterprise, the new code page converter file is copied into the `$ICU_DATA/icudt48<platform-suffix>` directory, where the number 48 denotes the version of ICU (4.8) that IBM App Connect Enterprise uses.

   To check if the level of ICU that is used by DFDL is different to that used by IBM App Connect Enterprise, complete one of the following tasks for your platform:

   a) Linux UNIX

   On Linux and UNIX system, look under the following path to find a library name that starts with `libicudata*`:

   ```
   <product installation path>/dfdlc/lib
   ```

   The numerical suffix to the file name denotes the ICU version.

   b) Windows

   On Windows systems, look under the following path to find libraries with names that start with `icudt*`:

   ```
   <product installation path>\bin
   ```

   Any libraries that have a numerical suffix that is not 48 are the DFDL ICU libraries, and the suffix denotes the ICU version.

2. If the ICU version that is used by the DFDL component within IBM App Connect Enterprise is not 48, a new subdirectory that is named *icudt<DFDL-icu-version><platform-suffix>* must be created under the $ICU_DATAdirectory.
   For example: If ICU libraries with a suffix of 51 are found, then the new code page converter file for DFDL must be copied to *$ICU_DATA/icudt51<platform-suffix>* directory, where the *<platform-suffix>* is one of the following values:

   - `l` for little-endian ASCII platforms
   - b for big-endian ASCII platforms
   - e for EBCDIC platforms

Chapter 5. Administering IBM App Connect Enterprise

Choose the method you prefer to activate and manage your integration nodes, integration servers, and associated resources.

About this task

Administration of IBM App Connect Enterprise includes some or all of the following tasks, depending on your particular implementation, where you are running App Connect Enterprise (on premises or in the cloud), and the features that you are using:

- “Managing integration nodes” on page 243
- “Managing integration servers” on page 249
- “Managing resources” on page 271
- “Managing deployed resources” on page 316
- “Overriding properties at run time with policies” on page 324
- “Viewing administration activity in the admin log” on page 330
- “Administering Java applications” on page 473
- “Changing the location of the IBM App Connect Enterprise working directory ” on page 474

These methods can be performed by using one or more of the administrative techniques supported by IBM App Connect Enterprise:

- The IBM App Connect Enterprise Toolkit
- The IBM App Connect Enterprise commands
- The IBM Integration API
- The Administration REST API
- The IBM App Connect Enterprise web user interface

For each task, the administrative techniques that you can use are identified.

Managing integration nodes

Work with your existing integration nodes to manage their connections and their active status by using the IBM App Connect Enterprise Toolkit, the IBM Integration API, or the command line.

About this task

- “Connecting to an integration node by using the Toolkit” on page 245

  Connect to a remote integration node from the IBM App Connect Enterprise Toolkit.
- “Connecting to an integration node by creating a .broker file” on page 244

  Export connection details of an integration node to a `.broker` file, so that another user or another integration node can use them.
- “Starting and stopping an integration node” on page 247

  Start and stop an integration node from the IBM App Connect Enterprise Toolkit or by using the command line.
- “Deleting an integration node” on page 165

  Delete an integration node from the IBM App Connect Enterprise Toolkit or by using the command line.

You can also use the IBM Integration API to complete some of these actions. See "Managing resources by using the IBM Integration API" on page 444.

To learn more about creating and configuring integration nodes, see the topics in "Configuring App Connect Enterprise to run on premises (on a physical or virtual machine)" on page 110.

Connecting to an integration node by creating a `.broker` file

View the schema and an example of a `.broker` file, which contains connection settings for a specific integration node.

Use a `.broker` file to define connection details for a specific integration node, including host, port, and optionally password and SSL security credentials. You can then share this stored connection information between different users and commands so that this information does not have to be typed manually each time. For example, you might specify a `.broker` file so that multiple users can connect to a remote integration node that has SSL enabled on its web administration port.

You can create a `.broker` file in a text editor by using the XML schema that is provided in this topic. You can reference the `.broker` file in the IBM App Connect Enterprise Toolkit, by using IBM App Connect Enterprise commands, and in the IBM Integration API.

You can also use a `.broker` file to define connection details for an independent integration server (which is an integration server that is not managed by an integration node).

The following **mqsi*** commands can reference a `.broker` file, by using the **-n** parameter:

- **mqsicacheadmin**
- **mqsichangeflowmonitoring**
- **mqsichangeflowstats**
- **mqsichangeresourcestats**
- **mqsicreateexecutiongroup**
- **mqsicredentials**
- **mqsideleteexecutiongroup**
- **mqsideploy**
- **mqsilist**
- **mqsipushapis**
- **mqsireloadsecurity**
- **mqsireportflowmonitoring**
- **mqsireportflowstats**
- **mqsireportresourcestats**
- **mqsistartmsgflow**
- **mqsistopmsgflow**

A sample `.broker` file is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<IntegrationNodeConnectionParameters Version="11.0.0" host="localhost" listenerPort="4414"
 integrationNodeName="INODE" userName="user"  password="password" useSsl="true"
 sslIncludeProtocols="TLSv1.2  "sslTrustStorePassword="password" sslTrustStorePath="c:\keystore
\my_keystore.jks" xmlns="http://www.ibm.com/xmlns/prod/ace/11" />
```

You can specify either `IntegrationNodeConnectionParameters` or `broker` as the root element in the `.broker` file. In the following example, the root element name is `IntegrationNodeConnectionParameters`.

This example enables an SSL connection to an integration node named INODE that is running on host 'localhost'. INODE uses port 4414 for its HTTPS web administration connections.

The `.broker` file is based on the following XML schema:

```
<?xml version="1.0" encoding="UTF-8"?><xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.ibm.com/xmlns/prod/ace/11">
    <xsd:element name="IntegrationNodeConnectionParameters">
     <xsd:complexType>
      <xsd:attribute name="Version" use="required">
       <xsd:simpleType>
        <xsd:restriction base="xsd:string">
         <xsd:pattern value="\d\d[0-9\.]*"></xsd:pattern>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="host" type="xsd:string"
       use="optional">
      </xsd:attribute>
      <xsd:attribute name="listenerPort" type="xsd:int"
       use="optional">
      </xsd:attribute>
      <xsd:attribute name="integrationNodeName" type="xsd:string"
       use="optional">
      </xsd:attribute>
      <xsd:attribute name="userName" type="xsd:string"
       use="optional">
      </xsd:attribute>
                  <xsd:attribute name="password" type="xsd:string"
       use="optional">
      </xsd:attribute>
                        <xsd:attribute name="useSsl" type="xsd:boolean"
      use="optional">
      </xsd:attribute>
                 <xsd:attribute name="sslIncludeProtocols" type="xsd:string"
       use="optional">
      </xsd:attribute>
      <xsd:attribute name="sslTrustStorePassword" type="xsd:string"
       use="optional">
      </xsd:attribute>
      <xsd:attribute name="sslTrustStorePath" type="xsd:string"
       use="optional">
      </xsd:attribute>
     </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Connecting to an integration node by using the Toolkit

Before you can administer an integration node by using the IBM App Connect Enterprise Toolkit, you must connect to the integration node.

Before you begin

- Create an integration node

Before you can connect to an integration node from the Toolkit, the integration node and its web user interface HTTP connection listener must be running. You can check that the listener was started by using the **mqsilist** command or by checking for the BIP3132 message in the integration node system log.

About this task

You can create a connection to an integration node by using the Integration Explorer view in the IBM App Connect Enterprise Toolkit. You can create the connection by using either the connection wizard or settings from a `.broker` file.

By default, when you create a connection to an integration node, the Toolkit connects to it automatically when the Toolkit is started. However, if you want to create the connection details without automatically connecting the Toolkit to the integration node, you can turn off the automatic connection setting by clicking **Window** > **Preferences** > **Integration Development** > **Connection Settings**, and then deselecting **Automatic connection to remote Integration Nodes and Integration Servers**. Then click **Apply and Close**.

Procedure

Create a connection to an integration node by using one of the following methods:

- Create a connection to an integration node by specifying the hostname and port settings in the connection wizard:
  a) Open the Integration Explorer view in the IBM App Connect Enterprise Toolkit.
  b) Right-click the Integration Nodes folder, and click **Connect to an Integration Node**.

     A wizard guides you through the steps involved in connecting to your integration node. As a minimum, you must specify the hostname and port. If administration security has been enabled on the integration node, you must specify authentication details (username and password) or select the **Use HTTPS** option.
  c) In the **Connect to an integration node** wizard, enter the following values:
     - The **Host name** or IP address of the computer on which your integration node is running.
     - The **Port** used by the administration REST API. This property must be a valid positive number.
     - (Optional). The **User name** for a web administration account that you want to use to secure the connection. For more information, see command.
     - (Optional). The **Password** that is associated with the web administration account for the **User name**. For more information, see command and "Managing web user accounts" on page 2510.
     - To save the **Password** value, select the **Save password** checkbox.

       The password is encrypted in Eclipse Secure Storage, which is a local file that is shared by all Eclipse-based products on the computer, including the IBM App Connect Enterprise Toolkit. If you select this option and Eclipse Secure Storage is not already set up, you must complete some additional steps to store the password securely:

       a. Set a master password for Eclipse Secure Storage.

          On Windows, the master password is generated automatically.

          On Linux and UNIX, the **Secure Storage** window opens, and you must set a master password for Eclipse Secure Storage. Enter a value for **Password**, and use the same value for **Confirm password**.

       b. On all distributed systems, you are prompted to set master password recovery.

          If you select **Yes**, the **Password Recovery** pane opens. In the **Password Recovery** pane, enter your questions and answers for password recovery.
     - To connect to the integration node by using HTTP with the Secure Sockets Layer (SSL), select the **Use HTTPS** checkbox.
  d) Click **Finish** to connect to the integration node.
- Create a connection to an integration node by using settings from a `.broker` file:
  a) Open the Integration Explorer view in the IBM App Connect Enterprise Toolkit.
  b) Right-click the Integration Nodes folder, and click **Connect to an Integration Node using an integration connection file**.
  c) Select the `.broker` file and then click **Open** to connect to the integration node.

  The connection to the remote integration node is created in your workspace.

Results

You can now view properties and manage your integration node, and its integration servers and resources, in the IBM App Connect Enterprise Toolkit.

You can export the remote connection details to a `.broker` file by right-clicking the integration node and selecting **Export the Integration Node to an Integration Connection file**.

What to do next

You can use the toolkit to create integration servers for the integration node, as described in Creating an integration server by using the IBM App Connect Enterprise Toolkit.

You can delete or change the value of a saved password for the connection. To change or delete the saved password:

1. Right-click the integration node that has a saved password.
2. Select **Change Saved Connection Password**.
3. Enter a new password, or leave the field empty to delete the saved password.

Starting and stopping an integration node

Run the appropriate command to start or stop an integration node.

Before you begin

- Ensure that your user ID has the correct authorizations to perform the task. Refer to Security requirements for administrative tasks.
- On Linux, UNIX, and Windows systems, you must set up your command line environment before you perform this task, by running the product profile or console; see “Setting up a command environment” on page 64.

About this task

To start and stop an integration node, you can use the commands from the command line.

Follow the links for instructions for the appropriate operating system:

Procedure

- “Starting and stopping an integration node on Windows, Linux, and UNIX systems” on page 247

Starting and stopping an integration node on Windows, Linux, and UNIX systems

Use the **mqsistart** or **mqsistop** commands to start or stop an integration node.

Before you begin

- Ensure that your user ID has the correct authorizations to perform the task. Refer to Security requirements for administrative tasks.
- On Linux, UNIX, and Windows systems, you must set up your command prompt environment before you perform this task, by running the product profile or console; see “Setting up a command environment” on page 64.
- When you start an integration node, all integration servers under that node are started automatically unless one of the following conditions applies:
  1. The integration node was stopped by using the **mqsistop** command, the Enterprise Toolkit, or the web user interface.
  2. You specified that an integration server must not be started. To specify that an integration server must not be started automatically, add an empty file that is named `.stopped` into the server's directory; for example:

     ```
     C:\myACEwork\components\NODEIPL101\servers\ISDEVIPLAA\.stopped
     ```
- If you use the **mqsistop** command or the Enterprise Toolkit or web user interface to stop an integration server, a `.stopped` file is added to the server's directory. When the integration node is next restarted,

the integration server is not started automatically (unless the `.stopped` file is deleted before the integration node is started).

Procedure

- Optional: On Windows, if the integration node is configured to use an IBM App Connect Enterprise vault, copy the `.mqsivaultrc` file into the directory that is referenced by the environment variable *MQSI_WORKPATH*. The default value for *MQSI_WORKPATH* is `C:\ProgramData\IBM\MQSI`.

  For more information about IBM App Connect Enterprise vaults and the `.mqsivaultrc` file, see command - systems and command.
- To start an integration node, enter the following command on the command line:

  ```
  mqsistart INODE
  ```

  Substitute your own integration node name for INODE.

  To start an associated queue manager for the integration node, use the **strmqm** command. For more information, see the topic *Starting and stopping a queue manager* in the IBM MQ documentation.

  The integration node is started. If you are using Windows, the command initiates the Windows service for the integration node.

  You can check that the integration node is running by using the **mqsilist** command; for example:

  ```
  mqsilist
  ...
  BIP1325I: Integration node 'NODEIPL101' with administration URI 'http://<host>:4417' is
   running.
  ```

  You can also check that the integration node initialized successfully by reviewing the following logs:

  – On Windows, check the Application Log in the Event Viewer.

  – On Linux and UNIX, check the system log.

  The log contains messages about verification procedures. If all tests are successful, only an initial start message is recorded. If one or more verification tests are unsuccessful, the log also includes messages that provide details of the tests that failed. If errors are reported, review the messages, and take the suggested actions to resolve these problems.
- To stop an integration node, enter the following command on the command line:

  ```
  mqsistop INODE
  ```

  Substitute your own integration node name for INODE.

  To stop the associated queue manager, run the **endmqm** command. For more information, see the topic *Starting and stopping a queue manager* in the IBM MQ documentation.

  You can check that the integration node is stopped by using the **mqsilist** command; for example:

  ```
  mqsilist
  ...
  BIP1326I: Integration node 'ACE02NODE' is stopped.
  ```

What to do next

For more information about verifying the status of an integration node, see “Verifying an integration node” on page 120.

Viewing integration node properties

You can view integration node properties by looking at the contents of the `node.conf.yaml` configuration file for the integration node.

Procedure

1. Use a YAML editor to open the `node.conf.yaml` configuration file for the integration node, which is created automatically when you create the integration node.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, if you modify the contents of the file, you must ensure that you do not include any tab characters, which are not valid in YAML.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Optional: If you want to modify any of the integration node properties, follow the instructions in "Configuring an integration node by modifying the node.conf.yaml file" on page 118.

Managing integration servers

Manage your integration servers by using the IBM App Connect Enterprise Toolkit, the web user interface, or the command line. You can also use the IBM App Connect Enterprise REST API or the IBM Integration API to complete some of these actions.

About this task

The IntegrationServer process is a process that is similar to other operating system processes. You can view how much CPU or memory it is consuming, by using standard operating system tools and commands.

You can see what applications, REST APIs, integration services, and shared libraries are deployed to the integration server, by using the Integration Explorer view in the IBM App Connect Enterprise Toolkit, the IBM App Connect Enterprise web user interface, or the REST API.

For more information about integration servers, see Integration servers and integration nodes.

Complete the steps in the following topics to start and stop an integration server, and connect to an integration server from the IBM App Connect Enterprise Toolkit:

- "Starting an integration server" on page 250
- "Stopping an integration server" on page 254
- "Connecting to an integration server by using the Toolkit" on page 255
- "Connecting to an independent integration server from the Toolkit by specifying a userid and password" on page 256

For more information about creating, configuring, and deleting an integration server, see the following topics:

- "Creating an integration server" on page 168
- "Configuring an integration server by modifying the server.conf.yaml file" on page 172
- "Deleting an integration server" on page 175

Starting an integration server

You can start an integration server and all the resources deployed to it by using the IBM App Connect Enterprise Toolkit, the web user interface, the **IntegrationServer** command, or the administration REST API.

About this task

If resources are deployed to your integration server, the started or stopped state of the resources is retained when you next start the integration server.

To start an integration server, you can use the following methods:

- "Starting an integration server that is managed by an integration node, by using the IBM App Connect Enterprise Toolkit" on page 250
- "Starting an integration server that is managed by an integration node, by using the web user interface" on page 251
- "Starting an independent integration server by using the IntegrationServer command" on page 251
- "Starting a managed integration server by using the mqsistartmsgflow command" on page 253
- "Starting an integration server by using the administration REST API" on page 338

You can also create and start a local, independent integration server by using the IBM App Connect Enterprise Toolkit, as described in "Creating and starting a local, independent integration server by using the Toolkit" on page 170.

Starting an integration server that is managed by an integration node, by using the IBM App Connect Enterprise Toolkit

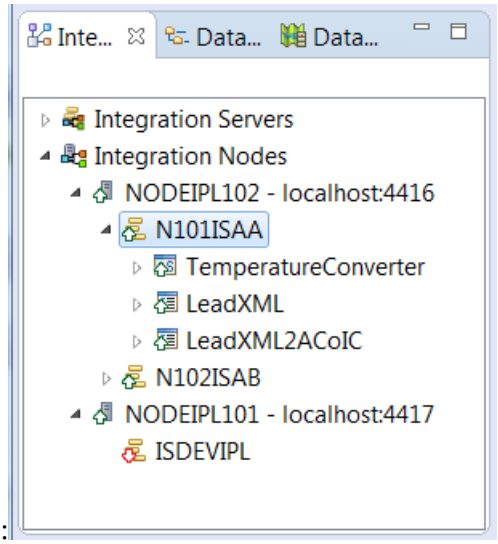
Procedure

To start an integration server under an integration node by using the IBM App Connect Enterprise Toolkit, complete the following steps.

1. Open the IBM App Connect Enterprise Toolkit, and switch to the Integration Development perspective.
2. In the Integration Explorer view, expand the integration node, right-click the integration server, and then click **Start**.

Results

The selected integration server is started. The Integration Explorer view is updated to indicate that the server and resources are running; for example:



Starting an integration server that is managed by an integration node, by using the web user interface

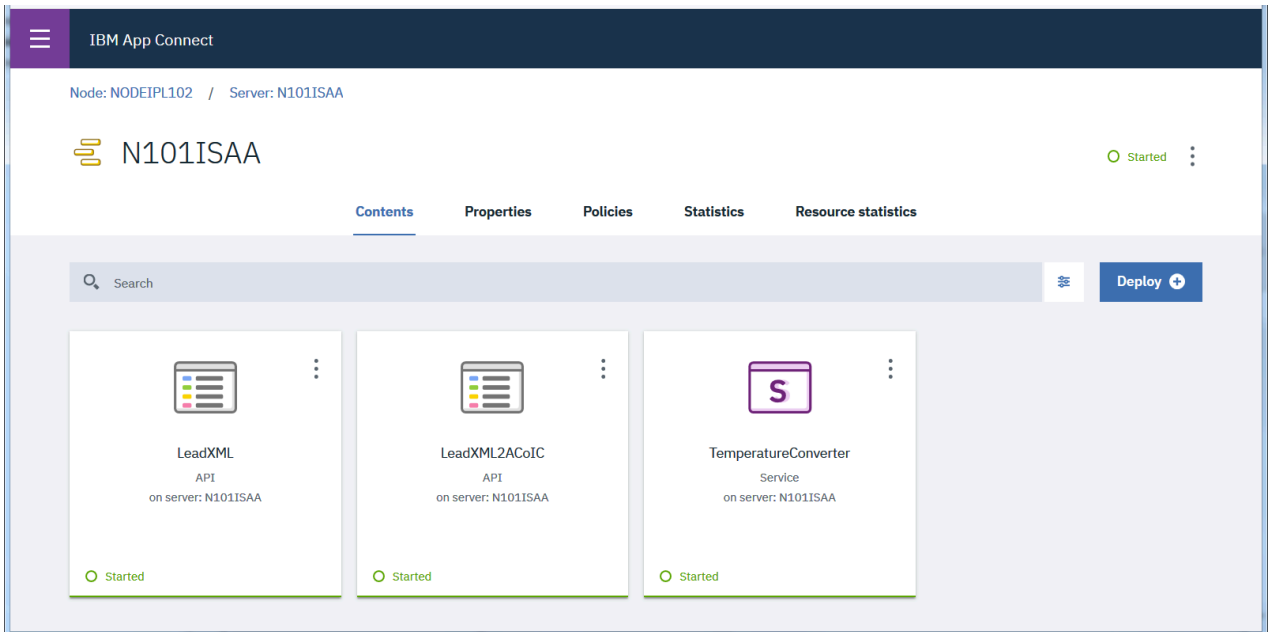
Procedure

To start an integration server under an integration node by using the web user interface, complete the following steps.

1. Start the web user interface for your integration node, as described in “Accessing the web user interface” on page 89.
2. Find the integration server that you want to start, select it, and then click **Start**.

Results

The selected integration server is started, along with the resources deployed to that server. The web user interface is updated to indicate that the server and resources are running; for example:



Starting an independent integration server by using the `IntegrationServer` command

You can use the **`IntegrationServer`** command to start an independent integration server (an integration server that is not managed by an integration node).

Before you begin

- Ensure that you have set up your command environment, as described in “Setting up a command environment” on page 64.
- Create a work directory and configure the integration server, as described in “Configuring an integration server by modifying the server.conf.yaml file” on page 172.

Procedure

To start an independent integration server, complete the following steps:

1. Run the **IntegrationServer** command, specifying the path to the work directory of the integration server.
   For example:

   ```
   IntegrationServer --work-dir c:\mywrk\myaceworkdir
   ```

   where *c:\mywrk\myaceworkdir* is the integration work directory that you created when you configured the integration server.

   For more information about creating the work directory, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172 and command.

   If the integration server contains any independent resources (resources that are not in an application or library), you must specify the **--default-application-name** parameter the first time that you run the **IntegrationServer** command for this integration server. This parameter creates a new default application, and all independent resources that are associated with the integration server are moved into it. For example:

   ```
   IntegrationServer --work-dir c:\mywrk\myaceworkdir --default-application-name myDefaultApp
   ```

   For more information, see command.

   If you have created a vault in your work directory, you must supply the correct vault key when starting the integration server. To supply the vault key, you can specify either the **--vault-key** or **--vaultrc-location** parameters on the **IntegrationServer** command, or you can set the MQSI_VAULT_KEY or MQSI_VAULTRC_LOCATION environment variables. If you specify none of these, the `.mqsivaultrc` file will be looked for in your HOME directory. If you are using the `.mqsivaultrc` file, it must be configured to contain the correct vault key. For more information about using the vault, see "Configuring encrypted security credentials" on page 177.

   If you have created integration tests for flows on the integration server, you can run those tests by using the **--test-project** parameter on the **IntegrationServer** command to specify the name of the integration test project that contains the tests. For example:

   ```
   IntegrationServer –work-dir c:\tmp\work-dir --test-project MyIntegrationTestProject
   ```

   where the **--work-dir** parameter specifies the work directory of the integration server that will run the tests, and the **--test-project** specifies the name of the test project to be run. The tests in the specified test project run automatically when the integration server starts. For more information, see "Running tests by using the command line" on page 1932.

   When you run the **IntegrationServer** command, the integration server starts, using the configuration that is defined in its `server.conf.yaml` file and any overrides that have been set by commands. When you run commands that modify an integration server, an overrides directory is created under the working directory for the integration server, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
2. You can now interact with the running integration server, by using the IBM App Connect Enterprise Toolkit or the web user interface.

   You can also investigate the administrative REST API provided by IBM App Connect Enterprise. For example, you might want to verify that the integration server is running, by sending a REST GET request to port 7600 and checking to see if there is a response:

   ```
   curl -X GET http://localhost:7600/apiv2
   ```

   If you want to use the curl tool, you might need to download and install it in addition to IBM App Connect Enterprise.

   The REST administration port, which is the primary method of communicating with the integration server, is set by default to 7600 (through the **adminRestApiPort** property in the `server.conf.yaml` configuration file).

3. Optional: Use the IBM App Connect Enterprise Toolkit to connect to the integration server:
   a) In the **Integration Explorer** view, right-click **Integration servers**, and then click **Connect to an integration server**. In the Connection details dialog, enter the host name and port for the integration server. Ensure that the port matches the value of the **admin-rest-api** property specified on the **IntegrationServer** command or the **RestAdminListener / port** property that was specified in the `server.conf.yaml` file (in this example, the port is `7600`, and the host name is `localhost`). If the integration server is secured, you must also specify the user name and password.
   b) Click **Finish**.

      The connection for your integration server is now displayed in the **Integration Explorer** view in the toolkit.
   c) You can now deploy a BAR file (or an application or REST API) by dragging it from the Application Development view and dropping it onto the integration server in the Integration Explorer view. For more information about deployment, see "Deploying integration solutions during development" on page 2464 and "Deploying integration solutions to a production environment" on page 2480.
   d) You can view the properties of the integration server, by selecting it in the Integration Explorer view, and then clicking the **Properties** tab. This shows the properties that were specified for the integration server in the `server.conf.yaml` configuration file, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Starting a managed integration server by using the `mqsistartmsgflow` command

Procedure

You can use the **mqsistartmsgflow** command to start an integration server that is managed by an integration node, by completing the following steps:

1. Open a command line for your current installation.
2. Enter the **mqsistartmsgflow** command, specifying the parameters for the integration server that you want to start.
   - If the integration node is local, specify the integration node name. For example:

     ```
     mqsistartmsgflow INODE -e default
     ```
   - If the integration node is remote, you can specify a configuration file. For example:

     ```
     mqsistartmsgflow -n INODE.broker -e default
     ```
   - If the integration node is remote, you can alternatively specify the connection parameters **-i** and **-p**. For example:

     ```
     mqsistartmsgflow -i 9.20.193.11 -p 4414 -e default
     ```
   - If you want to start all the integration servers on an integration node, specify the **--all-integration-servers** flag instead of **-e**.For example:

     ```
     mqsistartmsgflow INODE --all-integration-servers
     ```

   See the **mqsistartmsgflow** command description for more details about these options.

What to do next

Start or stop resources that are deployed on your integration server; see "Starting or stopping deployed resources" on page 317.

Stopping an integration server

You can stop an integration server under an integration node by using the IBM App Connect Enterprise Toolkit, the web user interface, or the administration REST API. You can stop an independent integration server by stopping its IntegrationServer process.

About this task

If resources are deployed to your integration server, the started or stopped state of the applications, integration services, and message flows is remembered when you stop the integration server. You can start or stop deployed resources before the integration server is stopped; see “Starting or stopping deployed resources” on page 317.

For an integration server under an integration node, the integration node processes all inflight messages and associated transactions for each message flow before it stops.

To stop an integration server under an integration node, you can use one of the following methods:

- “Stopping an integration node's server by using the IBM App Connect Enterprise Toolkit” on page 254
- “Stopping an integration server by using the web user interface” on page 254
- “Stopping an integration server by using the administration REST API” on page 338

Tip: If you use the Enterprise Toolkit or web user interface to stop an integration server under an integration node, a `.stopped` file is added to the server's directory. When the integration node is next restarted, the integration server will not be restarted automatically (unless the `.stopped` file is deleted before the integration node is started).

To stop an independent integration server by following these steps:

- **On Windows**, you can stop an integration server either by stopping the IntegrationServer process (for example, in the Windows Task Manager), or by pressing **Ctrl+C** in the App Connect Enterprise command console.
- **On Linux**, you can stop the integration server by sending the SIGINT signal to the IntegrationServer process.

Stopping an integration node's server by using the IBM App Connect Enterprise Toolkit

Procedure

To stop an integration server under an integration node by using the IBM App Connect Enterprise Toolkit, complete the following steps:

1. Open the IBM App Connect Enterprise Toolkit, and switch to the Integration Development perspective.
2. In the Integration Explorer view, expand the integration node, right-click the integration server, and then click **Stop**.

Results

The selected integration server is stopped, and a `.stopped` file is added to the server's directory. The Integration Explorer view is updated to indicate that the server is stopped.

Stopping an integration server by using the web user interface

Procedure

To stop an integration server by using the web user interface:

1. Start the web user interface for your integration node, as described in “Accessing the web user interface” on page 89.

2. Expand the list of folders to find the one that you want to stop, right-click it, and then click **Stop**.

Results

The selected integration server is stopped, and a `.stopped` file is added to the server's directory. The web user interface is updated to indicate that the server is stopped.

Connecting to an integration server by using the Toolkit

Before you can administer an integration server by using the IBM App Connect Enterprise Toolkit, you must connect to the integration server.

Before you begin

- Create an integration server

About this task

You can create a connection to an integration server by using the Integration Explorer view of the IBM App Connect Enterprise Toolkit. You can create the connection by using either the connection wizard or settings from a `.broker` file.

By default, when you create a connection to an integration server, the Toolkit connects to it automatically when the Toolkit is started. However, if you want to create the connection details without automatically connecting the Toolkit to the integration server, you can turn off the automatic connection setting by clicking **Window** > **Preferences** > **Integration Development** > **Connection Settings**, and then deselecting **Automatic connection to remote Integration Nodes and Integration Servers**. Then click **Apply and Close**.

Procedure

Create a connection to an integration server by using one of the following methods:

- Create a connection to an integration server by specifying the hostname and port settings in the connection wizard:
  a) Open the Integration Explorer view in the IBM App Connect Enterprise Toolkit.
  b) Right-click the Integration Servers folder and then click **Connect to an Integration Server**.
  c) In the **Connect to an integration server** wizard, enter the following values:
     – The **Hostname** or IP address of the computer on which your integration server is running.
     – The **Port** used by the administration REST API. This property must be a valid positive number.
     – (Optional). The **User name** for a web administration account that you want to use to secure the connection. For more information, see command.
     – (Optional). The **Password** that is associated with the web administration account for the **User name**. For more information, see command and “Managing web user accounts” on page 2510.
     – To save the **Password** value, select the **Save password** checkbox.

       The password is encrypted in Eclipse Secure Storage, which is a local file that is shared by all Eclipse-based products on the computer, including the IBM App Connect Enterprise Toolkit. If

you select this option and Eclipse Secure Storage is not already set up, you must complete some additional steps to store the password securely:

a. Set a master password for Eclipse Secure Storage.

On Windows, the master password is generated automatically.

On Linux and UNIX, the **Secure Storage** window opens, and you must set a master password for Eclipse Secure Storage. Enter a value for **Password**, and use the same value for **Confirm password**.

b. You are prompted to set master password recovery.

If you select **Yes**, the **Password Recovery** pane opens. In the **Password Recovery** pane, enter your questions and answers for password recovery.

– To connect to the integration server by using HTTP with the Secure Sockets Layer (SSL), select the **Use HTTPS** checkbox.

d) Click **Finish** to connect to the integration server.

- Create a connection to an integration server by using settings from a `.broker` file:
  a) Open the Integration Explorer view in the IBM App Connect Enterprise Toolkit.
  b) Right-click the Integration Servers folder and then click **Connect to an Integration Server using an Integration Connection file**.
  c) Select the `.broker` file and then click **Open** to connect to the integration server.

  The connection to the integration server is created in your workspace.

Results

You can now view properties and manage your integration server and resources, in the IBM App Connect Enterprise Toolkit.

You can export the remote connection details to a `.broker` file by right-clicking the integration server and selecting **Export the Integration Server to an Integration Connection file**.

Connecting to an independent integration server from the Toolkit by specifying a userid and password

Connect to an independent integration server from the IBM App Connect Enterprise Toolkit by specifying a userid and password.

Before you begin

- Create an independent integration server by running the **`IntegrationServer`** command as described in command, or by using a wizard in the IBM App Connect Enterprise Toolkit, as described in “Creating and starting a local, independent integration server by using the Toolkit” on page 170.

About this task

You can configure an independent integration server to require a userid and password to be specified when a user tries to connect to it from the IBM App Connect Enterprise Toolkit.

Procedure

Configure the independent integration server with a userid and password by completing the following steps:

1. Stop the independent integration server by stopping its IntegrationServer process as described in “Stopping an integration server” on page 254.

2. Enable authentication on the independent integration server by running the **mqsichangeauthmode** command.

   For example, if the work directory for your independent integration server is `myWorkDir`, you can run the following command:

   ```
   mqsichangeauthmode -w myWorkDir -b active
   ```

   For more information, see command.
3. Create a web user account and password for an independent integration server by running the **mqsiwebuseradmin** command.

   For example, if the work directory for your independent integration server is `myWorkDir`, you can run the following command to create the web user *Admin* with the password *Passw0rd!*:

   ```
   mqsiwebuseradmin -w myWorkDir -c -u Admin -a Passw0rd!
   ```

   For more information, see mqsiwebuseradmin command.
4. Start the independent integration server by running the **IntegrationServer** command.

   For example, if the work directory for your independent integration server is `myWorkDir`, you can run the following command:

   ```
   IntegrationServer --work-dir myWorkDir
   ```

   For more information, see command.

Connect to the independent integration server from the IBM App Connect Enterprise Toolkit by completing the following steps:

5. In the Integration Explorer view, right-click **Integration servers** and then click **Connect to an integration server**.
6. In the **Connection details** wizard, enter the following information:

   - **Host name**

     Enter the name or the URL of the host where the independent integration server is running. For example, `localhost`.

   - **Port**

     Specify the port number through which the REST API is used for integration server administration. For example, `7600`.

   - **User name**

     Specify user name for the web user account that you created. For example, `Admin`.

   - **Password**

     Specify password for the web user account that you created. For example, `Passw0rd!`.

What to do next

Start or stop resources that are deployed on your integration server. For more information, see “Starting or stopping deployed resources” on page 317.

Viewing integrations by using an IBM App Connect Dashboard

Use an IBM App Connect Dashboard to view local and remote integrations, and integrations that are running on containers.

About this task

You can start an IBM App Connect Dashboard by running the **Dashboard** command, and logging in to it in a web browser. You can then use it to create connections to independent and managed integration servers.

Multiple users can log on to the same IBM App Connect Dashboard. You can view the connections that you create under your account, and you can export them to be viewed by other users in the same dashboard, or in a different dashboard. You can import and view connections that other users create in the same dashboard, or in a different dashboard. When you export connections, you can encrypt credentials by specifying a password, which other users must use when they import your connections.

By selecting the appropriate tab, you can view the state of all integration nodes, or integration servers that have connections in an IBM App Connect Dashboard, and that your account is authorized to view. By traversing the menus, you can view information about any resource that is deployed to any of the integration servers.

Create an IBM App Connect Dashboard, configure integration servers to connect to it, create and share connections, and view information as described in the following topics:

Procedure

- “Starting an IBM App Connect Dashboard by using the Dashboard command” on page 258
- “Logging in and out of an IBM App Connect Dashboard” on page 260
- “Configuring an integration server to connect to an IBM App Connect Dashboard” on page 261
- “Creating connections in an IBM App Connect Dashboard” on page 264

View integrations in an IBM App Connect Dashboard, as described in the following topics:

- “Viewing integration servers that are connected to an IBM App Connect Dashboard ” on page 266
- “Viewing the state of integration servers that are connected to an IBM App Connect Dashboard ” on page 268
- “Viewing information from deployed resources in an IBM App Connect Dashboard ” on page 269

Share connection information with other users in an IBM App Connect Dashboard, as described in the following topic:

- “Sharing connection information with other users in an IBM App Connect Dashboard ” on page 270

Starting an IBM App Connect Dashboard by using the Dashboard command

Start an IBM App Connect Dashboard by using the **Dashboard** command.

Procedure

Start an IBM App Connect Dashboard by using the **Dashboard** command.

You must specify a value for the **--work-dir** parameter. You can create a work directory before you run the command. If the work directory that you specify does not exist, it is created when you run the command.

You can include the **--port-number** parameter in the command. If you do not, the command creates a `dashboard.conf.yaml` file in the work directory with the port number set to the default value of 7700. You can allocate a non-default port by using one of the following methods:

- Before you run the **Dashboard** command, create a work directory and edit the `dashboard.conf.yaml` to reference a non-default port number. Then, run the **Dashboard** command without the **--port-number** parameter.
- Run the **Dashboard** command, and specify a non-default value for the **--port-number** parameter.

If the default port number is in use, or if the port number that you specify is in use, an error message appears in the command output.

If you supply a keystore, you must edit the **sslCertificate** parameter in the **--work-dir**`\dashboard.conf.yaml` file to show the path to the keystore file. You can supply the keystore password by using one of the following methods:

- Before you run the **Dashboard** command, create a work directory and edit the **sslPassword** parameter in **--work-dir**`\dashboard.conf.yaml` to show your password. Then, run the **Dashboard** command without the **--ssl-password** parameter.
- If you prefer not to specify the password in the **--work-dir**`\dashboard.conf.yaml` file, you can run the **Dashboard** command with the **--ssl-password** parameter. For more information, see command.

```
Connections:
  Https:
    # Reject connections to TLS endpoints presenting a certificate not signed by a trusted CA
 (i.e. self-signed certificates)
    rejectUnauthorized: true

RestAdminListener:
  port: 7700               # Set the Admin REST API Port for ACE Web UI and Toolkit. Defaults
 to 7700

  # Note the Admin REST API will be insecure without the following being set
  host: 'localhost'         # Set the hostname otherwise we bind to the unspecified address

  # SSL Server auth
  # sslCertificate: '/path/to/myKeystore.p12'
  # sslPassword: 'passw0rd'

  # SSL Client auth
  # requireClientCert: true                    # Request a certificate from the client
  # caPath: '/path/to/CA/certificates'         # CA certs, all files at this path will be read
```

For example, to start the dashboard, and use a work directory of `C:\myDashboardWorkDir`, and accept the default port, run the following command:

```
Dashboard --work-dir C:\myDashboardWorkDir
```

To start the dashboard, and use a work directory of `C:\myDashboardWorkDir1`, and specify a non-default port, run the following command. In this example, a value of 7701 is used for **--port-number**:

```
Dashboard --work-dir C:\myDashboardWorkDir1 --port-number 7701
```

To start the dashboard, and use a work directory of `C:\myDashboardWorkDir2`, and specify a non-default port, and specify an *ssl* password, run the following command:

```
Dashboard --work-dir C:\myDashboardWorkDir2 --port-number 7702 --ssl-password mySSLPassword
```

In this example, a value of 7702 was used for **--port-number**, and a value of *mySSLPassword* was used for **--ssl-password**.

To use the command with the **--ssl-password** parameter, you must configure the `--work-dir \dashboard.conf.yaml` file with a value for the **sslCertificate** parameter. You can configure the `dashboard.conf.yaml` file after you run the command. Alternatively, before you run the command, you can create the work directory `C:\myDashboardWorkDir2`, and copy the provided sample `dashboard.conf.yaml` file into the directory. Then, configure the `dashboard.conf.yaml` file with values for *sslCertificate*, and *sslPassword* before you run the command.

The sample `dashboard.conf.yaml` file is located in the `samples\configuration` folder of the installation directory. For example, `C:\Program Files\IBM\ACE\11.0.0.11\server\sample\configuration`,

Results
The dashboard process starts.

What to do next
Log in to the IBM App Connect Dashboard, as described "Logging in and out of an IBM App Connect Dashboard" on page 260.

Logging in and out of an IBM App Connect Dashboard
Log in to an IBM App Connect Dashboard by creating a new account, or by logging in with an existing account, and enable multiple users on the same dashboard. Log out of an IBM App Connect Dashboard by selecting **log out** from the **Help menu**.

Before you begin
Start an IBM App Connect Dashboard as described in "Starting an IBM App Connect Dashboard by using the Dashboard command" on page 258.

About this task
You can log in to the IBM App Connect Dashboard by creating a new account, or by logging in with an existing account, and you can enable multiple users to access the same dashboard. You can log out by selecting **Log out** from the **Help menu**.

Procedure
1. Start the IBM App Connect Dashboard in a web browser by using the following method:
   - From a web browser, enter the URL for the web user interface into a web browser, in the following form:

     protocol`://`*hostname*`:`*port*
     - *protocol* is either `http` or `https`.
     - *hostname* is the host that contains the work directory that you created by running the **`Dashboard`** command, as described in "Starting an IBM App Connect Dashboard by using the Dashboard command" on page 258.
     - *port* is the port on which the dashboard listens for requests. The default value is 7700.

     For example, if the dashboard is running on your local workstation on port 7700, you can enter the following URL:

     ```
     https://localhost:7700
     ```

     A web browser window opens, and the logon screen for IBM App Connect Dashboard is displayed.

Log in to the IBM App Connect Dashboard by creating a new account, as described in Step 2, or by logging in with an existing account, as described in Step 3. Multiple users can log in to the same dashboard at the same time by completing either of these steps.

2. Log in to the IBM App Connect Dashboard by creating a new account, by completing the following steps:
   a) Click **Create a new account**. The `Save password` dialog is displayed.
   b) In the `Username` field, type a value for the new username that you want to create.
   c) In the `Password` field, type a value for a password for the new username that you created.
   d) In the `Confirm password` field, type the value of the password that you created.
   e) Click **Create and log in**.

   The IBM App Connect Dashboard is opened, and the Dashboard view is displayed.
3. Log in to the IBM App Connect Dashboard by logging in to an existing account, by completing the following steps:
   a) In the `Username` field, type the username for an existing account.
   b) In the `Password` field, type the password for the username that you entered.
   c) Click **Login**.

   The IBM App Connect Dashboard is opened, and the Dashboard view is displayed.

When you finish working on the IBM App Connect Dashboard, you can log out by completing step 4

4. Optional: Log out of the IBM App Connect Dashboard by clicking **Log out** from the **Help menu**. The dashboard closes, and the logon screen is displayed.

What to do next

Create and configure integration servers to connect to an IBM App Connect Dashboard, as described in "Configuring an integration server to connect to an IBM App Connect Dashboard" on page 261.

Configuring an integration server to connect to an IBM App Connect Dashboard

Configure an integration server to connect to an IBM App Connect Dashboard.

About this task

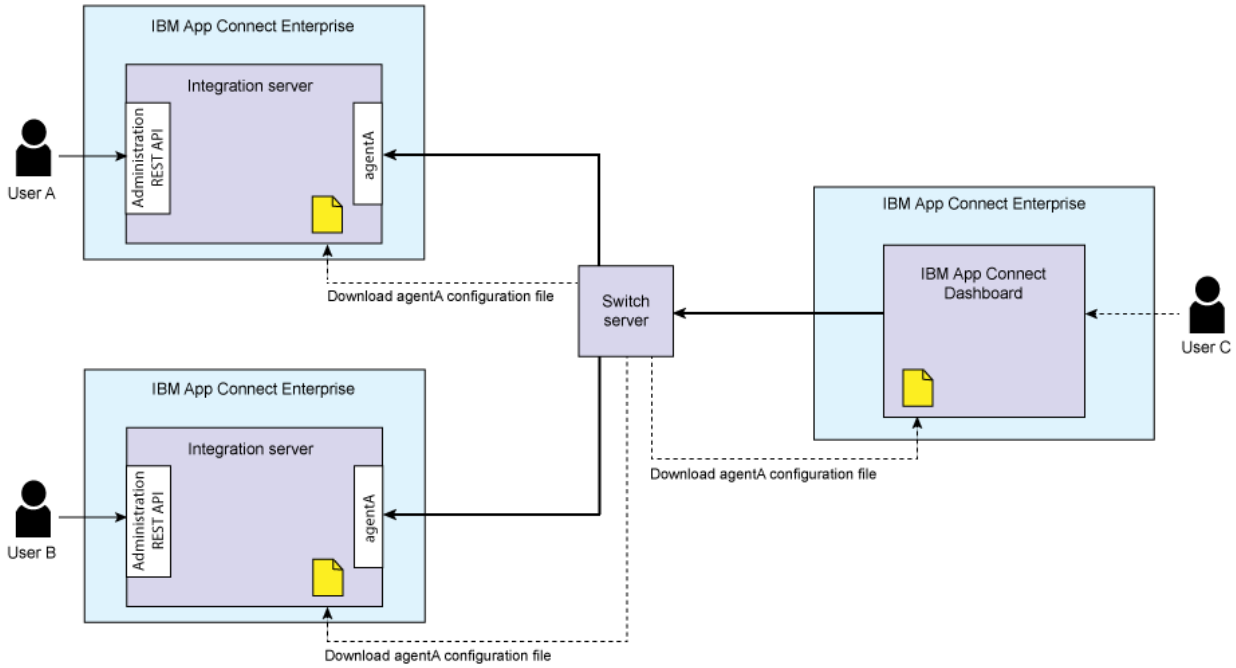
If you start an integration node, you can visualize the managed integration servers in the dashboard by creating a connection to the integration node from the dashboard. The only way that you can visualize a managed integration server in the dashboard is by creating a connection to the integration node in the dashboard.

If you start an independent integration server, you can visualize it in the dashboard by creating a connection to it from the dashboard.

If you configure an independent integration server with an `agentA` configuration file for a switch, it registers with the switch. Then, you can create a connection to the switch in the dashboard to visualize the independent integration server.

If you configure an independent integration server with an `agentA` configuration file, other users who can download the `agentA` file from the switch server can perform remote administration on your integration server from an IBM App Connect Dashboard. Any users and roles that are created for securing the administrative RESTAPI, are not applicable to remote administration with an `agentA` file. For more information, see "Managing resources by using the administration REST API" on page 334.

As shown in the diagram, if multiple integration servers are connected to a single switch server, remote administration can be performed on the connected integration servers from an IBM App Connect Dashboard. In this example, access for users A and B is controlled separately through the administration REST API of the respective Integration Servers. However, user C can perform remote administration on both integration servers by using the switch server and the `agentA` configuration file.



You cannot configure an integration node with an `agentA` configuration file. You can configure only independent servers with an `agentA` configuration file.

Procedure

If an integration server has a publicly accessible end point, you can connect directly to it from the IBM App Connect Dashboard, as described in step 1.

If an integration server is behind a firewall, you must configure the integration server with an `agentA` file, and then connect to it from the IBM App Connect Dashboard by using a switch server. You can obtain an `agentA` file by using a managed service, or by creating a switch operand in IBM Cloud Pak for Integration, as described in step 2.

If you want to connect to an integration server that is running on a container, you must also configure it with an `agentA` file.

1. Connect directly to an integration server that has a publicly accessible end point, as described in “Creating connections in an IBM App Connect Dashboard” on page 264.

If you configure an independent integration server with an `agentA` configuration file for a switch, it registers with the switch. Then, you can create a connection to the switch in the dashboard to visualize the independent integration server.

2. Configure the integration server with an `agentA` file by using a managed service, as described in substep a, or by creating a *SwitchServer* operand in IBM App Connect Operator, as described in substep b.
   a) Configure the integration server with an `agentA` file by using a managed service by completing the following steps:
      i) Download the `agentA` file by completing the following steps:
         a) If you have not created an App Connect Agent, click **New** on the **Private Network Connections** tab in App Connect, and then click **App Connect Agent**.
         b) On the **Actions** menu for the App Connect Agent, select **View setup instructions** to open a window in which the instructions are displayed.
         c) Download the configuration by clicking the button under step 2 of the instructions, which are displayed in the window, which you opened in step 2.
      ii) Copy the downloaded `agenta.json` configuration file to the directory `--work-dir/config/iibswitch/agenta`, where `--work-dir` is the work directory that you specified when you created the integration server. You might need to create the `agenta` directory at the same level as the `agentx` directory that is there by default. You cannot configure `agenta` and `agentx` on the same integration server.
   b) Create an `agentA` file as described in App Connect Switch Server reference. Then, configure the integration server with the `agentA` file by creating a SwitchServer operand in IBM App Connect Operator version 1.0.0 or later (operand 11.0.0.10-r3-eus) by running the following commands. The parameter values that are shown are examples only.
      i) `oc get Configuration my-switch-agenta -o=jsonpath={.spec.secretName} my-switch-agenta`
      ii) `oc get secret my-switch-agenta -o=jsonpath={.data.configuration} eyJuYW1lIjoiYWdlbnRhIiwic3dpdGNoIjp7InVybCI6IndzczovL21hcnRpbi1zcy1zd2l0Y2gtYWNlL0tLVxuIl0sInJlamVjdFVuYXV0aG9yaXplZCI6dHJ1ZX19fQ==`
      iii) `oc get secret my-switch-agenta -o=jsonpath={.data.configuration} | base64 -d | jq`

      The output from running the commands is similar to the following example:

```
{
  "name": "agenta",
  "switch": {
    "url": "wss://example.ibm.com:443",
    "certs": {
      "key": "-----BEGIN PRIVATE KEY-----\nMIIEvwIBADANBgk ... bNo12hvjjI2NJBwVIA==
\n-----END PRIVATE KEY-----\n",
      "cert": "-----BEGIN CERTIFICATE-----\nMIID8jCCAtqgAwIB ... Zu/Ep+k/\n-----END
 CERTIFICATE-----\n",
      "ca": [
        "-----BEGIN CERTIFICATE-----\nMIIEXjCCA0agA ... HpL0V1+U7xrajhpjh+tbTmD6\n-----
END CERTIFICATE-----\n"
      ],
      "rejectUnauthorized": true
    }
  }
```

```
}
```

iv) Save the output to a local file, for example, `/my/work/dir/config/iibswitch/agenta/agenta.json`.

v) Edit `/my/work/dir/config/iibswitch/agenta/agenta.json` by adding an **agentID** field at the same level as the **name** field.

Alternatively, you can set the *MQSI_AGENT_ID* environment variable in the environment for the independent integration server.

The value for **agentID** must be a *uuidv4* string.

The following example shows an `agenta.json` file that was edited to add an **agentID** field that is populated with a *uuidv4* string.

```
{
  "name": "agenta",
  "agentId": "7690095b-3db8-40d2-8584-d09335a0dab9"
  "switch": {
    "url": "wss://example.ibm.com:443",
    "certs": {
      "key": "-----BEGIN PRIVATE KEY-----\nMIIEvwIBADANBgk ... bNo12hvjjI2NJBwVIA==
\n-----END PRIVATE KEY-----\n",
      "cert": "-----BEGIN CERTIFICATE-----\nMIID8jCCAtqgAwIB ... Zu/Ep+k/\n-----END
 CERTIFICATE-----\n",
      "ca": [
        "-----BEGIN CERTIFICATE-----\nMIIEXjCCA0agA ... HpL0V1+U7xrajhpjh+tbTmD6\n-----
END CERTIFICATE-----\n"
      ],
      "rejectUnauthorized": true
    }
  }
}
```

vi) Copy the edited `agenta.json` configuration file to the directory `--work-dir/config/iibswitch/agenta`, where `--work-dir` is the work directory that you specified when you created the integration server. You might need to create the `agenta` directory at the same level as the `agentx` directory that is there by default. You cannot configure `agenta` and `agentx` on the same integration server.

What to do next

Create connections in an IBM App Connect Dashboard, as described "Creating connections in an IBM App Connect Dashboard" on page 264.

Creating connections in an IBM App Connect Dashboard

Create connections in an IBM App Connect Dashboard.

Before you begin

Start an IBM App Connect Dashboard by using the **Dashboard** command, and access it in a web browser, as described in "Starting an IBM App Connect Dashboard by using the Dashboard command" on page 258.

About this task

In the IBM App Connect Dashboard, you can create connections to managed integration servers and independent integration servers that are created with IBM App Connect Enterprise 11.0 Fix Pack 11 or later.

The Switch Server must be IBM App Connect Enterprise 11.0 Fix Pack 11 or later, or be a Switch Server that was created from an App Connect deployment in a containerized environment. For example, IBM App Connect Operator version 1.0.0 or later (operand 11.0.0.10-r3-eus). For more information, see Installing the IBM App Connect Operator.

Alternatively, if you have an instance of the service for App Connect on IBM Cloud, you can use the Switch Server that is created for you. For more information, see App Connect.

Procedure

1. In the IBM App Connect Dashboard, click **Deployment locations** to go the Deployment locations view.
2. Click **Create connection** to open the **Create connection** dialog box.
3. Click in the **Select a connection type** field to open the menu.
4. Select one of the following connection types:
   - Switch
   - Independent server
   - Integration node

   If you want to connect to a local integration node or a remote integration node, select **Integration node**.

   If you want to connect to a local independent integration server, or a remote independent integration server, select **Independent server**.

   If you want to connect to an integration server that runs behind a firewall, or in a container, select **Switch**.
5. Complete the **Name** field. If you want to include a description, you can complete the **Description** field for your connection.
   a) In the **Name** field, type a name for your new connection. If you are connecting to an integration node, or an independent integration server, the connection can have the same name as the integration node or integration server. Alternatively, you can give the connection any name that is not in use for a connection. Valid characters to use for a connection are A-Z, a-z, 0-9, -, and _.
   For example, if you want to create a connection to a local integration node named *myNode*, you can name the connection `myNode`.
   b) Optional: In the **Description** field, type a description.
   For example, `Local integration node`.

If you selected **Integration node** or **Integration server** at Step "4" on page 265, you must complete Step 6. If you selected **Switch** at Step "4" on page 265, you must complete Step 7.

6. Optional: Enter the connection values for an integration node, or an independent integration server.
   a) You must complete the **address** field. In the **address** field, type the URL for the local or remote integration node or the local or remote integration server with the format, *protocol://hostname:port*.
      - *protocol* is either `http` or `https`.
      - *hostname* is for the host of the target integration server you are trying to connect to.
      - *port* is the port (on the *hostname*) of the admin RESTAPI of the target integration node or integration server that you are trying to connect to.

      For example, `http://localhost:4414`
   b) Optional: If you enabled security, you must complete the **Username** field. Type the username that you authorized for access.
   c) Optional: If you enabled security, you must complete the **Password** field. Type the password for the username that you entered in substep b.
   d) If the **address** field URL *protocol* is `https`, you can complete the following fields. These fields are visible only when you input a URL with a *protocol* of `https` into the **address** field.
      - **PEM-formatted CA file**. If the server certificate is from a non-default CA, complete this field. Alternatively, you can update the `--work-dir/dashboard.conf.yaml` file parameter **`rejectUnauthorized`** to *false*.

```
Connections:
```

```
  Https:
    # Reject connections to TLS endpoints presenting a certificate not signed by a
trusted CA (i.e. self-signed certificates)
    rejectUnauthorized: false
```

- **PKCS12 keystore file**. If you need to provide a client certificate for client authentication, complete this field.
- **Keystore passphrase**. If you provide a *PKCS12 Keystore file*, complete this field.

e) In the **Create Connection** dialog box, click **Connect**. The new connection appears in the list of connections in the Deployment locations view.

7. Optional: If you are connecting to a switch server, you must import an `agentA` file. You can obtain an `agentA` file, as described in "Configuring an integration server to connect to an IBM App Connect Dashboard" on page 261.

   a. In the **Create connection** dialog box, click **Click to select an agentA file** in the **AgentA file** field to open a file explorer window.

   b. In the file explorer window, go to the directory where your `agentA` file is located, and select the file.

   c. In the file explorer window, click **Open** to select the file. The field **File imported** appears in the **Create connection** dialog box. The field is populated with the name of the file that you selected.

   d. In the **Create Connection** dialog box, click **Connect**. The new connection appears in the list of connections in the Deployment locations view.

   If you connect to an integration server by using a Switch Server, you must be aware of the properties of the integration server that you connect to.

   If you connect to an independent integration server that is running on a virtual machine, and the integration server is stateful, any changes that you make from the IBM App Connect Dashboard are persistent. If the integration server is stateless, the changes are not persistent, and they are lost if the virtual machine crashes.

   If the integration server is running in a container, then it was created declaratively, and has ephemeral storage. If you change the integration server from the IBM App Connect Dashboard, for example, by deploying a BAR file, that change is performed. However, if multiple replicas of the integration server exist, the change is only performed on one of the replicas. If the container crashes, the changes are lost.

What to do next

View connections in the IBM App Connect Dashboard, as described in "Viewing integration servers that are connected to an IBM App Connect Dashboard " on page 266.

Viewing integration servers that are connected to an IBM App Connect Dashboard

Use an IBM App Connect Dashboard to view managed integration servers, or independent integration servers.

Before you begin

Start an IBM App Connect Dashboard by using the **Dashboard** command, and access it in a web browser, as described in "Starting an IBM App Connect Dashboard by using the Dashboard command" on page 258.

Create connections in an IBM App Connect Dashboard, as described in "Creating connections in an IBM App Connect Dashboard" on page 264.

About this task

In the IBM App Connect Dashboard, you can view only the connections that you create under your account, or connections that other users share with your account. For more information, see “Sharing connection information with other users in an IBM App Connect Dashboard ” on page 270.

Procedure

1. In the IBM App Connect Dashboard, click the **Dashboard** icon to go to the Dashboard view.

You can view any of the following integration servers:

- Local or remote managed integration servers on IBM App Connect Enterprise.
- Local or remote independent integration servers on IBM App Connect Enterprise.
- Integration servers that are connected to the IBM App Connect Dashboard by a switch server.

View a connection in the IBM App Connect Dashboard by completing one of the following steps:

2. Optional: Local or remote managed integration server on IBM App Connect Enterprise. View a local or a remote managed integration server by completing the following steps:
   a) In the Dashboard view, click **Nodes**. Any local or remote integration nodes that are connected to the IBM App Connect Dashboard, and that your account is authorized to view, are displayed.

      The tile for each integration node contains the following information:

      - The name of the integration node.
      - The text "Node" to confirm that the connection is to an integration node.
      - The URL of the connection. For example, *http://localhost:4414*.

      You can search for a specific integration node by typing the name of the integration node into the **Search** field and pressing Enter. If an integration node with the specified name is connected to the IBM App Connect Dashboard, it is highlighted in the Dashboard view.

      For more information about connection names, see “Creating connections in an IBM App Connect Dashboard” on page 264.

   b) View the managed integration server by accessing the menu in the connection tile for the integration node. Click the **Open List of Options** icon in the tile of an integration node. Then, click Open in the menu.

      If the integration node has any managed integration servers, they are displayed in the Dashboard view.

      The tile for each managed integration server contains the following information:

      - The name of the managed integration server.
      - The text "Server" to confirm that the tile is for an integration server.
      - The status of the managed integration server is shown as *Started* or *Stopped*.

      You can search for a specific managed integration server by typing the name or part of the name of the integration server into the **Search** field and pressing Enter. If a managed integration server with the specified name exists on the integration node, it is highlighted in the Dashboard view.

      Alternatively, you can view manged integration servers by accessing the **Servers** tab, as described in step 3.

3. Optional: You can view local or remote independent integration servers, and local or remote managed integration servers by accessing the **Servers** tab. You can also view integration servers that are connected to the IBM App Connect Dashboard by a switch. View integration servers by completing the following steps:
   a) In the Dashboard view, click **Servers**. Integration servers that your account is authorized to view, are displayed. You can view local or remote independent integration servers, and local or

remote managed integration servers. If you have a connection to a switch, the Servers view shows integration servers that are connected at the other side of that switch server.

A tile for an independent integration server differs from a tile for a managed integration server. It includes the URL. The following information is included on the tile of an independent integration server:

- The name of the work directory for the integration server. If you specified a name for the independent integration server when you created it, the name is shown instead of the work directory.
- The text "Server" to confirm that the connection is to an integration server.
- The URL of the connection. For example, *http://localhost:7700*.

The tile for an integration server that is connected to the IBM App Connect Dashboard by a switch also contain the following details:

- If the server is replicated, the number of replications are shown.
- The text "connected via: *<name of switch server>*"

You can search for a specific independent integration server by typing the name of the work directory for the integration server. If you specified a name when you created the independent integration server, you can search by typing the name of the integration server. If an independent integration server with the specified work directory or name is connected to the IBM App Connect Dashboard, it is highlighted in the Dashboard view.

For more information about connection names, see “Creating connections in an IBM App Connect Dashboard” on page 264.

What to do next

View the state of the integration servers, as described in “Viewing the state of integration servers that are connected to an IBM App Connect Dashboard ” on page 268.

Viewing the state of integration servers that are connected to an IBM App Connect Dashboard

View the state of integration servers that are connected to an IBM App Connect Dashboard.

Before you begin

Start an IBM App Connect Dashboard by using the **Dashboard** command, and access it in a web browser, as described in “Starting an IBM App Connect Dashboard by using the Dashboard command” on page 258.

Create connections in an IBM App Connect Dashboard, as described in “Creating connections in an IBM App Connect Dashboard” on page 264.

About this task

You can view the state of local and remote managed integration servers, independent integration servers, or integration servers that are connected to the IBM App Connect Dashboard by a switch server. The integration servers must have a connection in the IBM App Connect Dashboard that your account is authorized to view. You can view the state of the integration servers by inspecting the connection tiles in the Dashboard view.

Procedure

1. Use the IBM App Connect Dashboard to view integration servers, as described in “Viewing integration servers that are connected to an IBM App Connect Dashboard ” on page 266.

2. Inspect the tile for the integration server. The status is shown as *Started* or *Stopped*.

   You cannot change the state of an integration node or an integration server in the IBM App Connect Dashboard. For more information about starting and stopping integration nodes and integration servers, see "Starting and stopping an integration node" on page 247, "Starting an integration server" on page 250, and "Stopping an integration server" on page 254.

What to do next

"Viewing information from deployed resources in an IBM App Connect Dashboard " on page 269.

Viewing information from deployed resources in an IBM App Connect Dashboard

View information from deployed resourced in an IBM App Connect Dashboard.

Before you begin

Start an IBM App Connect Dashboard by using the **Dashboard** command, and access it in a web browser, as described in "Starting an IBM App Connect Dashboard by using the Dashboard command" on page 258.

Create connections in an IBM App Connect Dashboard, as described in "Creating connections in an IBM App Connect Dashboard" on page 264.

About this task

You can view information from integration servers that are connected to the IBM App Connect Dashboard, and that your account is authorized to view. To view information from a specific integration server, complete step "1" on page 269. To view information from all the integration servers, complete step "2" on page 269.

Procedure

1. To view information from a specific integration server, for example, the details of a message flow that is deployed in an application, complete the following steps:
   a) In the Dashboard view of the IBM App Connect Dashboard, click **Servers**. Integration servers that are connected to the IBM App Connect Dashboard, and that your account is authorized to view, are displayed.
   b) Click the **Open List of Options** icon for an integration server. Then, click Open in the menu.

      If any applications are deployed to the integration server, they are displayed in the Dashboard view.
   c) Click the **Open List of Options** icon for an application. Then, click Open in the menu.

      If any message flows are contained in the deployed application, they are displayed in the Dashboard view.
   d) Click the **Open List of Options** icon for a message flow. Then, click Open in the menu.

      You can view information about the message flow by clicking the **Properties** tab or the **Activity log** tab.
2. To view information from all integration servers that are connected to the IBM App Connect Dashboard, and that your account is authorized to view, click **Integrations** in the Dashboard view of the IBM App Connect Dashboard.

   If any resources are deployed to any of the integration servers that your account is authorized to view, they are displayed in the Dashboard view. You can view information by clicking the **Open List of Options** icon on the tile of any resource that is displayed.

What to do next

You can share connection information with other users, as described in “Sharing connection information with other users in an IBM App Connect Dashboard ” on page 270.

Sharing connection information with other users in an IBM App Connect Dashboard

Sharing connection information with other users in an IBM App Connect Dashboard by exporting and importing configuration files.

About this task

You can export connection information from your IBM App Connect Dashboard by downloading it in the form of a `yaml` file. If you want to encrypt any credentials in the exported connections, you must specify a password. If you need to restore this set of connections, you can import the `yaml` file in the same dashboard, or in a different dashboard. Other users can also import the `yaml` file on the same dashboard, or on a different dashboard. If a password was set when the connections were exported, you must use the password to import the connections, unless the exported `yaml` file does not contain any encrypted credentials.

Procedure

1. Export connection information. You can export connection information by completing the following steps:
   a. In the Deployment locations view, click **Export**. The "Export Connections" dialog box opens.
   b. In the "Export Connections" dialog box, type a password in the `Password (optional)` field.

      If you provide a password, it is used to encrypt any credentials in your exported connections. It is possible to view the file, but not the credentials. You need the password to import a connections file that contains encrypted credentials. You do not need a password to import a connections file that does not contain encrypted credentials, even if a password was provided when the file was exported.
   c. If you want to cancel the export action, click **Cancel**.
   d. Click **Export** to export the connections.

      Your connections are downloaded as a YAML file into the `Downloads` folder in your local file system.

   For example, `C:\Users\<your user id>\Downloads`.

   If you need to restore this set of connections, you can import the file. Another user on the same dashboard can also import this file.
2. Import connection information.

   You can import connection information that you or another user export from the same dashboard, or from a different dashboard. If a password was set when the connections were exported, and the `yaml`

file contains encrypted credentials, you must use the password to import the connections. You can import connection information by completing the following steps:

a. In the Deployment locations view, click **Import**. The Import Connections dialog box opens.
b. In the Import Connections dialog box, click **Click to select a yaml file**. A File Explorer window opens.
c. In the **File Explorer** window, go the location where the `yaml` file that you want to import is located.
d. In the **File Explorer** window, select the `yaml` file and click **Open**. The `File imported` field in the Import Connections dialog box is populated with the name of the `yaml` file that you selected.
e. If the `yaml` file that you import contains encrypted credentials, you must enter the password that was used when the `yaml` file was exported in the `Password` field.
f. If you want to cancel the import of the file, click **Cancel**,
g. Click **Import**. The `yaml` file is imported into the `Downloads` folder in your local file system.

For example, `C:\Users\<your user id>\Downloads`.

Managing resources

Manage the resources used by integration servers and integration nodes.

About this task

Read the following topics for information about managing resources that are used by integration servers or integration nodes:

- “Managing data caching” on page 271
- “Listing database connections” on page 314
- “Quiescing a database” on page 314
- “Using a JDBC connection pool to manage database resources used by an integration server” on page 315
- “Configuring properties for TCP/IP” on page 229

For information about managing resources that have been deployed to an integration server, see “Managing deployed resources” on page 316.

Managing data caching

Store data that you want to reuse by using the embedded global cache or an external WebSphere eXtreme Scale grid.

About this task

The topics in this section describe the data caching capability that is provided by WebSphere eXtreme Scale. For information about other caching solutions, see the following topics:

- “Environment tree” on page 508
- “Long-lived variables” on page 1611

WebSphere eXtreme Scale provides IBM App Connect Enterprise with data caching capability.

Find out more about the embedded global cache and external WebSphere eXtreme Scale grids in the following topics:

- “Data caching overview” on page 272
- “Embedded global cache” on page 277
- “WebSphere eXtreme Scale grids” on page 278
- “Global cache scenario: Storing state for integrations” on page 273

- "Global cache scenario: Caching static data" on page 275
- "Data caching terminology" on page 281
- "Configuring the embedded global cache" on page 285
- "Global cache sample: Configuring one catalog and one container" on page 286
- "Global cache sample: Configuring one catalog and four containers" on page 288
- "Global cache sample: Configuring two catalogs and four containers" on page 294
- "Global cache sample: Configuring for high availability (multi-instance)" on page 300
- "Connecting to a WebSphere eXtreme Scale grid" on page 309
- "Enabling SSL for external WebSphere eXtreme Scale grids" on page 310
- "Migrating global cache configuration settings" on page 312
- "Monitoring the global cache" on page 313

Data caching overview

WebSphere eXtreme Scale provides IBM App Connect Enterprise with data caching capability.

The topics in this section describe the data caching capability that is provided by WebSphere eXtreme Scale. For information about other caching solutions, see the following topics:

- "Environment tree" on page 508
- "Long-lived variables" on page 1611

Introduction

A global cache is a repository for data that you want to reuse. For example, you can use a global cache in IBM MQ message flows to store correlation information for use beyond a specific message flow node, instance of a message flow, or integration server. The cache facilitates the sharing of data within an integration server and between integration servers, and eliminates the need for an alternative solution, such as a database. You can use one message flow node to store data in the global cache, then a second node (in the same message flow or a separate flow), can retrieve that data from the global cache.

You can use a message flow node to interact with the global cache. Interactions with the cache happen outside the message flow transaction, and are committed immediately. If an exception is thrown downstream of the node that interacts with the cache, the cache interactions are not rolled back.

Scenarios and tutorials

For illustrations of how you can use a global cache, see the following scenarios and tutorials:

- "Global cache scenario: Storing state for integrations" on page 273
- "Global cache scenario: Caching static data" on page 275
- Tutorial: Using a Mapping node to graphically access a Lookup table that is stored in the Global Cache; see the Tutorials Gallery in the IBM App Connect Enterprise Toolkit.

Embedded global cache or external WebSphere eXtreme Scale grid

You can use the global cache that is supplied with IBM App Connect Enterprise, and you can configure IBM App Connect Enterprise to connect to an external WebSphere eXtreme Scale grid. You can work with multiple remote grids, and the embedded grid, at the same time. For more information, see "Differences between the embedded global cache and an external WebSphere eXtreme Scale grid" on page 279.

For a detailed description of the elements involved in a global cache or external grid, see the following information:

- "Embedded global cache" on page 277
- "WebSphere eXtreme Scale grids" on page 278

Interaction with the global cache or external grid

You can interact with the global cache or external grid by using a Mapping node or a JavaCompute node. The node can put data into a map, retrieve data, and create a map if one does not exist. For more information, see Accessing the global cache by using a Mapping node or “Accessing the global cache by using a JavaCompute node” on page 1781.

Monitoring and administration of the global cache or external grid

You can monitor the global cache or external grid by using the following tools:

- The activity log
- Resource statistics

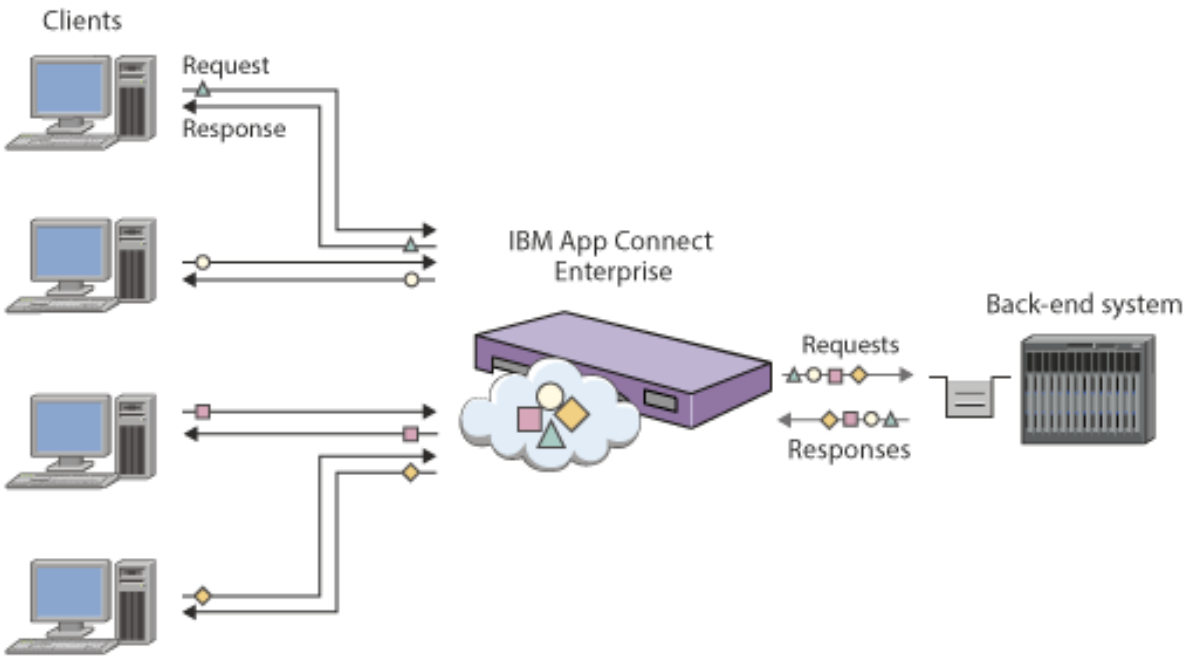
You can administer the embedded cache by using the **mqsicacheadmin** command.

For more information, see “Monitoring the global cache” on page 313.

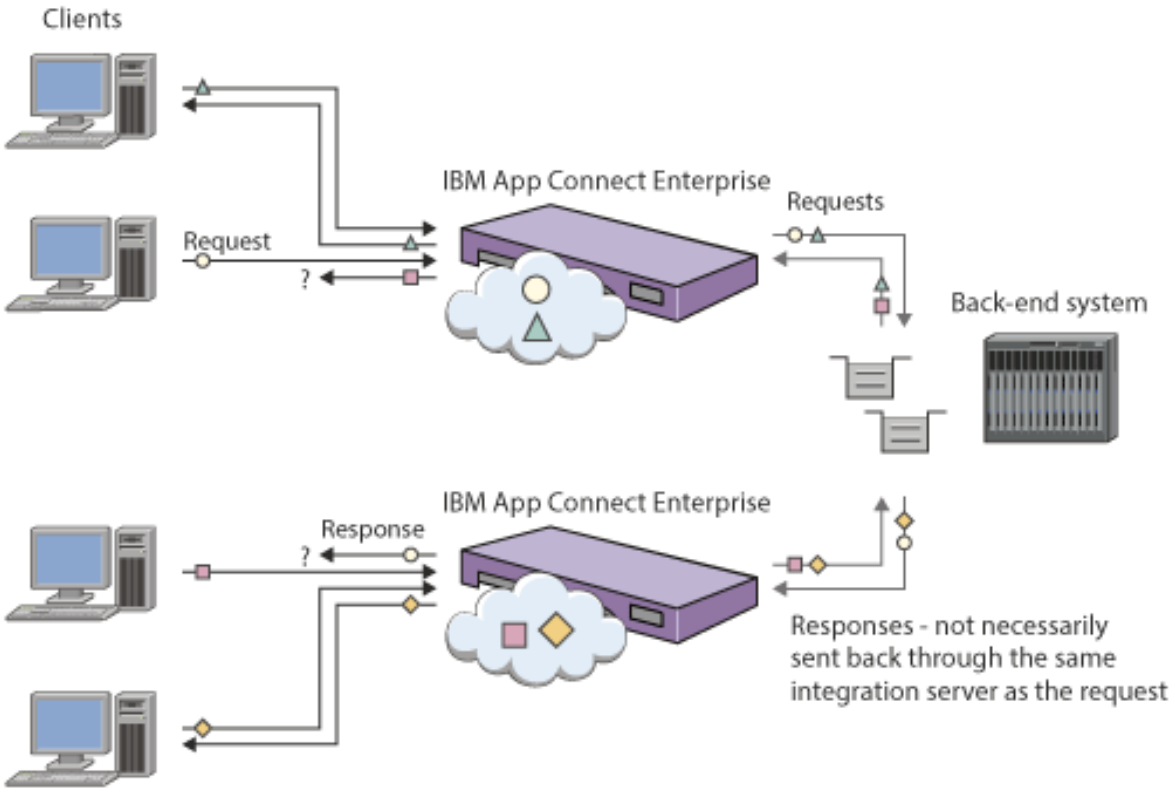
Global cache scenario: Storing state for integrations

You can use a global cache to store state for integrations. With a global cache, each integration server can handle replies, even when the request was processed by another integration server.

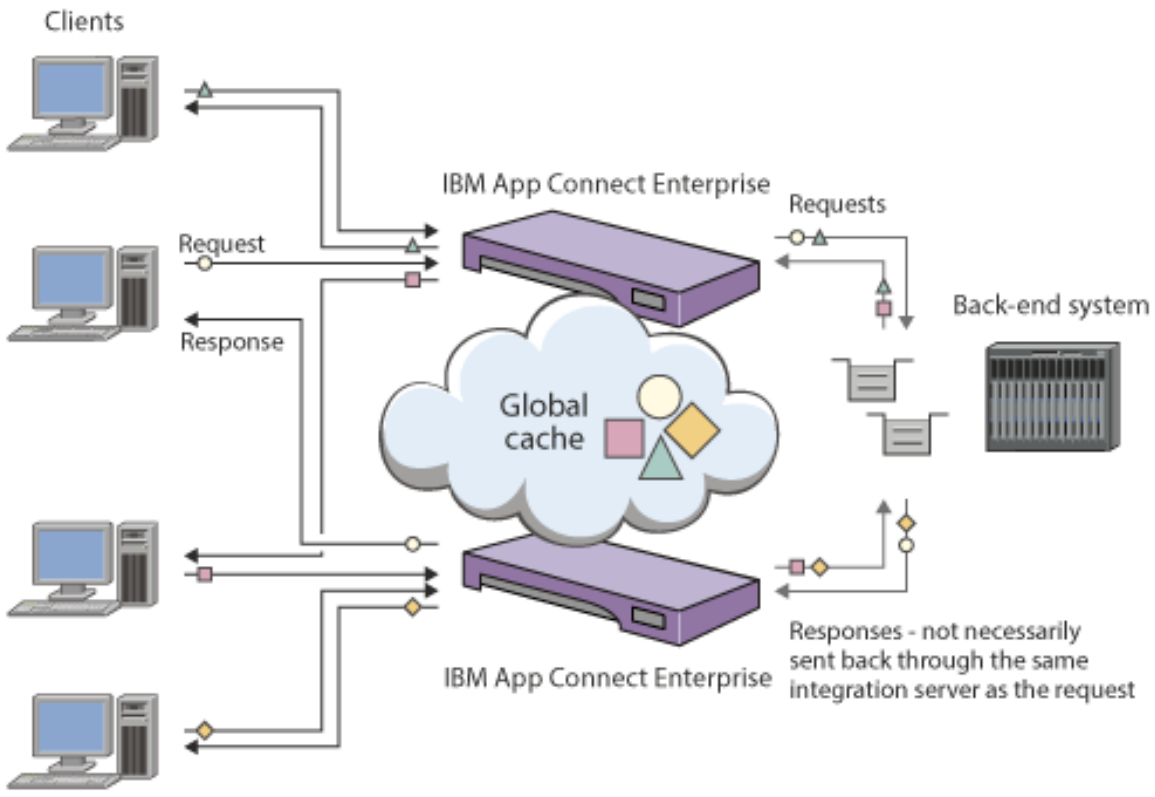
When IBM App Connect Enterprise is used to integrate asynchronous systems, the integration server records information about the requester to correlate the replies correctly, as shown in the following diagram.



When this integration is deployed to multiple integration servers for workload balancing, the reply will not necessarily return through the same integration server as the original request.



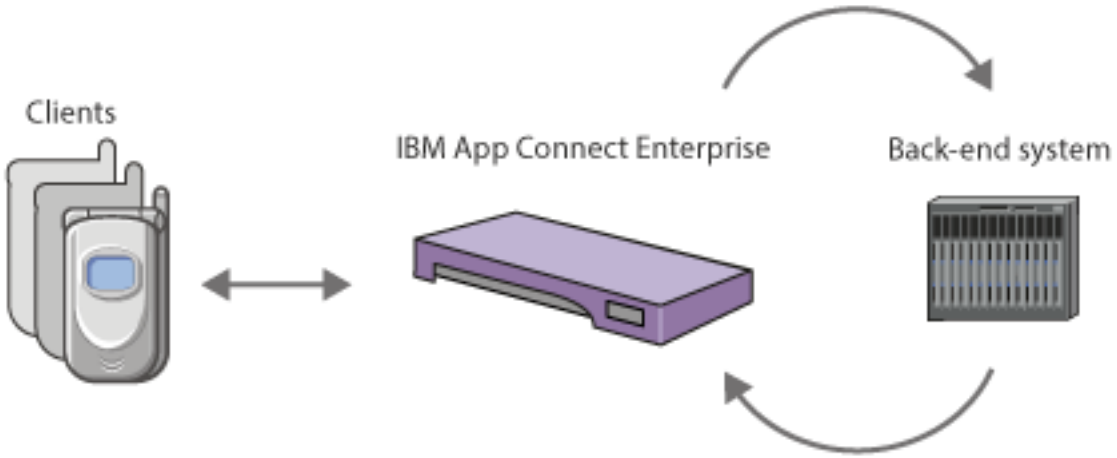
With a global cache, each integration server can handle replies, even when the request was processed by another integration server.



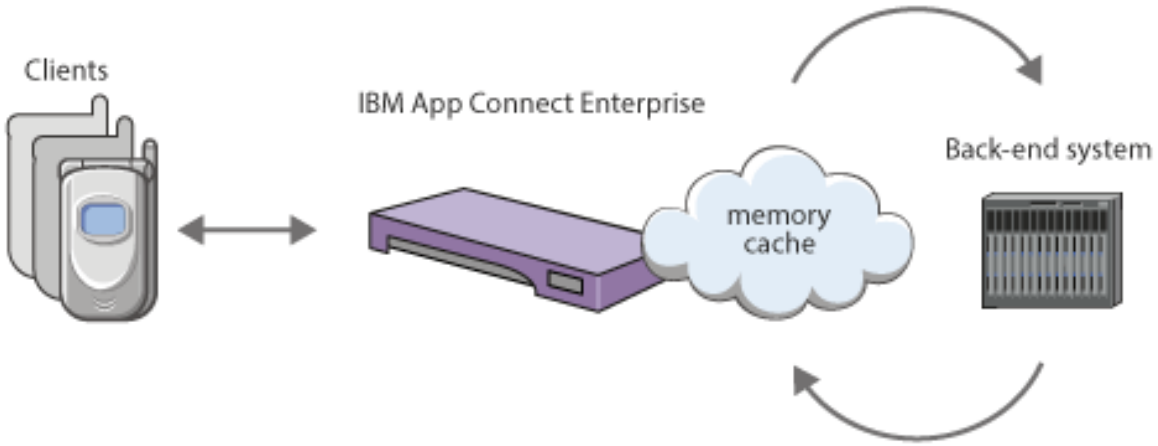
Global cache scenario: Caching static data

You can use a global cache to store static data. The use of a global cache facilitates horizontal scaling in situations where a cache is used to minimize network interactions to a back end system. With a global cache, you can increase the number of clients while maintaining a predictable response time for each client.

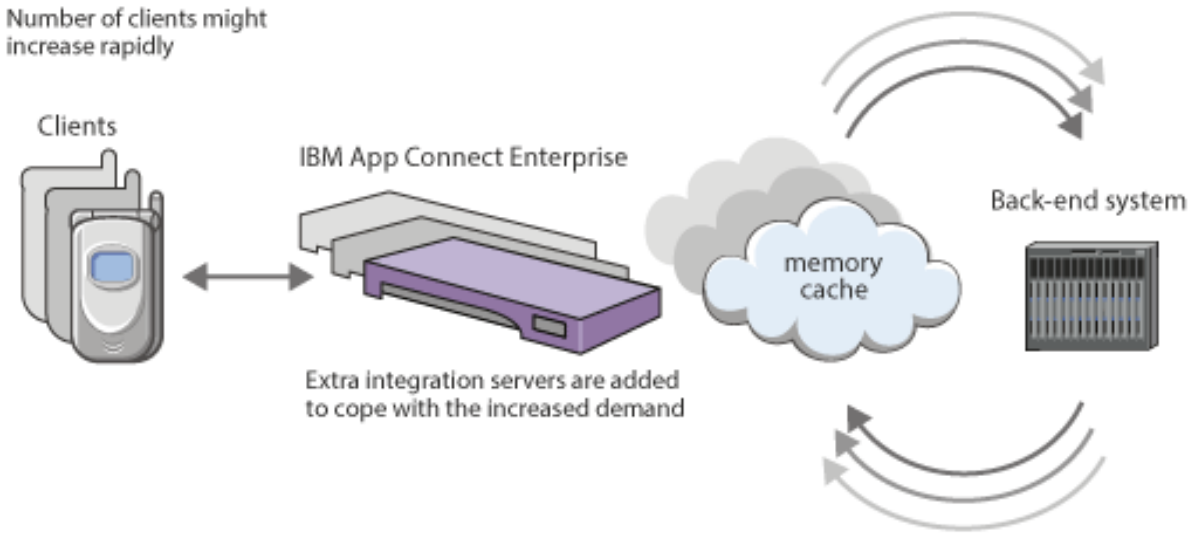
When IBM App Connect Enterprise acts as a façade to a back-end database, it must provide short response times to the client, even though the back-end database has high latency.



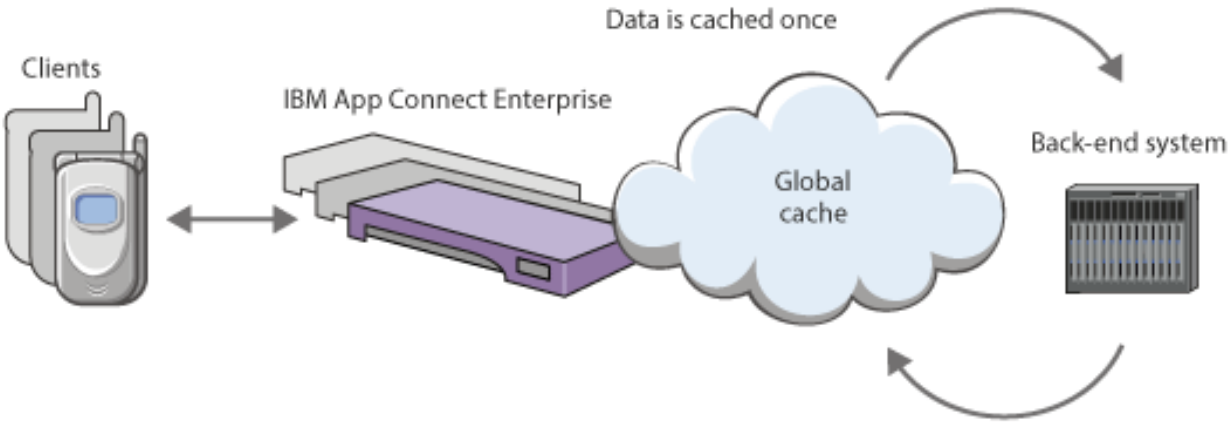
IBM App Connect Enterprise can provide short response times by caching results in memory (for example, ESQL shared variables).



However, this configuration does not scale well horizontally. When the number of clients increases, the number of integration servers can be increased to accommodate the clients, but each integration server has to maintain a separate in-memory cache.



With a global cache, the number of clients can increase while a predictable response time is maintained for each client.



Embedded global cache

Use the global cache that is supplied with IBM App Connect Enterprise to store data that you want to reuse.

The global cache is embedded in IBM App Connect Enterprise, and is disabled by default. To use the cache, you must enable and configure it by setting properties in the `server.conf.yaml` files for the integration servers that will share access to the cache.

The embedded global cache in IBM App Connect Enterprise can use a WebSphere eXtreme Scale enterprise data grid. Enterprise data grids use the eXtreme data format (XDF) instead of Java object serialization. XDF allows for class evolution, which means that you can evolve the class definitions that are used in the data grid without affecting older applications that are using previous versions of the class. For more information, see Enterprise data grid overview in the WebSphere eXtreme Scale product documentation.

Note: The App Connect Enterprise global cache is implemented using WebSphere eXtreme Scale technology. WebSphere eXtreme Scale is designed to offer a resilient cache topology by hosting a known number of catalog and container servers in a fixed topology. This is not compatible with the ephemeral nature of containers running in elastically-scaled cloud environments such as Kubernetes. Therefore, WebSphere eXtreme Scale does not support running the catalog or container server components in containerized environments such as Kubernetes, Docker, or Podman. As a result, App Connect Enterprise does not support the embedded cache when running an integration server in a containerized environment. The embedded cache is supported to make a client connection to an external WebSphere eXtreme Scale grid when running in a container.

To find out more about the differences between the embedded cache and the external grid, see “Differences between the embedded global cache and an external WebSphere eXtreme Scale grid” on page 279.

Customizing the embedded topology

You can set properties explicitly for each integration server, by setting properties in each `server.conf.yaml` configuration file. For example, you might want to specify particular integration servers to host the catalog and container servers so that you can tune performance.

If you stop the integration server that contains the catalog server, the cache becomes unavailable and the data in the cache is lost. Therefore, you must ensure that you define the catalog server appropriately. If you restart the integration server that hosts the catalog server, it can no longer communicate with the container servers in other integration servers. Although these container servers are still running, they are

no longer part of the cache, and your data is lost. Therefore, you must also restart the integration servers that host the container servers.

When you use multiple catalog servers, you can improve performance by taking the following steps:

- Provide other integration servers that host container servers only, rather than having only integration servers that host both catalog and container servers.
- Start and stop integration servers in sequence, rather than using the **mqsistart** or **mqsistop** commands to start or stop all integration servers at once. For example, start the integration servers that host catalog servers before you start the integration servers that host only container servers.

You can configure the global cache by using IBM App Connect Enterprise commands, or the IBM Integration API.

For more information, see "Configuring the embedded global cache" on page 285.

Interaction with the global cache

You can use Mapping nodes or JavaCompute nodes to store and retrieve data in a map in the global cache. When you get a global map from an external grid, the getGlobalMap method makes a connection to the grid if one does not exist. For more information, see Accessing the global cache by using a Mapping node or "Accessing the global cache by using a JavaCompute node" on page 1781.

When you get an MbGlobalMap object, you can also specify how long the data remains in the global cache before it is removed automatically. This time is known as the *time to live* and is counted from when that map entry is last updated. The value applies to all cache entries that are created by using that MbGlobalMap object in that instance of the message flow. Data that is already in the map that you specify, or that is created by another MbGlobalMap object, is not affected by the time to live value. You can create multiple MbGlobalMap objects in different flows or integration servers, all resolving to the same map in the global cache, but with different time to live values.

By default, the time to live is set to zero so that the data is never removed. To set a specific time to live, create a session policy, which you can reference from the MbGlobalMap object. For detailed instructions, see "Specifying how long data remains in the global cache by using the JavaCompute node" on page 1783.

WebSphere eXtreme Scale grids

Use one or more external WebSphere eXtreme Scale grids to store data that you want to reuse.

WebSphere eXtreme Scale provides a scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages data across multiple servers. The catalog servers and container servers for the IBM App Connect Enterprise global cache collaborate to act as a WebSphere eXtreme Scale grid.

For more information about the grid that is embedded in IBM App Connect Enterprise, see "Embedded global cache" on page 277. To find out more about the differences between the embedded cache and the external grid, see "Differences between the embedded global cache and an external WebSphere eXtreme Scale grid" on page 279.

In addition to the grid that is available in IBM App Connect Enterprise (as the embedded global cache), you can integrate with WebSphere eXtreme Scale grids that are running elsewhere. You can work with multiple external grids, and the embedded grid, at the same time. For a diagram that shows how IBM App Connect Enterprise message flows interact with an external WebSphere eXtreme Scale grid, and for definitions of the terms that are associated with a grid, see "Data caching terminology" on page 281. For more information about grids, see WebSphere Extreme Scale product documentation online.

You might have an existing WebSphere eXtreme Scale environment that uses a topology, such as one of the following examples:

- Embedded WebSphere eXtreme Scale servers in WebSphere Application Server network deployment installations
- A stand-alone WebSphere eXtreme Scale software grid

When you are connecting to an external grid, IBM App Connect Enterprise hosts the WebSphere eXtreme Scale client, but the cache is hosted on an external grid. Therefore, a separate WebSphere eXtreme Scale installation or appliance is required. An installation or appliance can host more than one grid.

To connect to an external grid, you need the following information:

- The name of the grid
- The host name and port of each catalog server for the grid
- Optional: the object grid file that can be used to override WebSphere eXtreme Scale properties

By using a policy to specify the parameters, you can connect to an external WebSphere eXtreme Scale grid. IBM App Connect Enterprise message flows can then access and modify data that is stored in the external grid. If you are connecting to a secure grid, you can create a security identity by using the **mqsisetdbparms** command. To connect to an external grid, follow the instructions in "Connecting to a WebSphere eXtreme Scale grid" on page 309.

You can enable SSL for client connections to external WebSphere eXtreme Scale grids by setting up a public key infrastructure, then enabling SSL for an integration server. For more details, see "Enabling SSL for external WebSphere eXtreme Scale grids" on page 310.

You can use Mapping nodes or JavaCompute nodes to store and retrieve data in a map in the global cache. When you get a global map from an external grid, the getGlobalMap method makes a connection to the grid if one does not exist. For more information, see Accessing the global cache by using a Mapping node or "Accessing the global cache by using a JavaCompute node" on page 1781.

You can monitor the external grid by viewing the activity log and resource statistics. For more information, see "Monitoring the global cache" on page 313.

Differences between the embedded global cache and an external WebSphere eXtreme Scale grid

The embedded cache is optimized for use in IBM App Connect Enterprise. When the default configuration that is provided by the embedded cache does not meet your needs, use an external WebSphere eXtreme Scale grid. With an external grid, you can also make the data in the cache available to both integration flows and other applications outside of IBM App Connect Enterprise.

You can use the embedded global cache or an external WebSphere eXtreme Scale grid (or both) in your IBM App Connect Enterprise solution.

The embedded global cache is a WebSphere eXtreme Scale grid, which consists of WebSphere eXtreme Scale catalog and container servers that are hosted in integration servers. This grid is designed and optimized for use within (and between) integration servers. You can configure the topology of this grid by specifying the number of catalogs and containers, and where they are hosted, but you cannot modify the underlying WebSphere eXtreme Scale configuration of the grid. The embedded cache is supplied as part of IBM App Connect Enterprise and no further installation is needed to use the cache. By contrast, the external grid is not provided with IBM App Connect Enterprise and must be obtained and installed separately. However, with an external grid, you have complete control over the WebSphere eXtreme Scale configuration.

The catalog and container servers for the embedded grid are hosted in integration server processes. Therefore, the catalog and container servers share their Java virtual machines (JVMs) with all other components that are running in the integration server. These components include IBM App Connect Enterprise components, user-defined Java code, and data. As a result, the cache servers do not have control over their containing JVMs, and cannot modify or restart the JVMs. The maximum JVM heap size is determined by the relevant integration server property. Also, the lifecycle of each cache server is tied to the lifecycle of the integration server in which it is hosted.

With an external grid, you must create, administer, and manage the WebSphere eXtreme Scale server components outside of IBM App Connect Enterprise. Therefore, you can separate the availability and management of your cache from IBM App Connect Enterprise.

For more information about connecting to external WebSphere eXtreme Scale grids, see "WebSphere eXtreme Scale grids" on page 278.

Common reasons for choosing an external grid over the embedded cache

Choose an external grid if you have any of the following requirements:

- You need to configure the grid for specific capabilities that are not supported by the embedded global cache.
- You have an architectural preference for the cached data not to be placed in the integration servers themselves.
- You want to separate the availability of the cache from the availability of the integration servers.
- You need an enterprise cache, with multiple applications (other than IBM App Connect Enterprise) that are accessing the data.
- You have (or want) sophisticated, or custom, tools to manage the cache.
- The cache needs to span multiple data centers for disaster recovery.

Comparison of function

The following table shows the requirements that are supported in the embedded global cache or an external grid:

Requirement	Embedded global cache	External WXS grid
A cache is available to IBM App Connect Enterprise with minimal configuration	√	
Cache components are hosted inside integration server processes	√	
Simple MbGlobalMap access exists to the cache	√	√
Access the global cache through the CacheGet, CachePut, and CacheRemove transforms in your maps	√	√
Same cached data can be accessed by multiple integration servers	√	√
Connections are managed by IBM App Connect Enterprise	√	√
Resource statistics and activity log provided for cache interactions	√	√
Manage your cache components outside of IBM App Connect Enterprise processes		√
Configure the number of partitions and the number and type of replicas		√
Configure multiple zones		√
Define your own backing maps, both fixed and templates		√

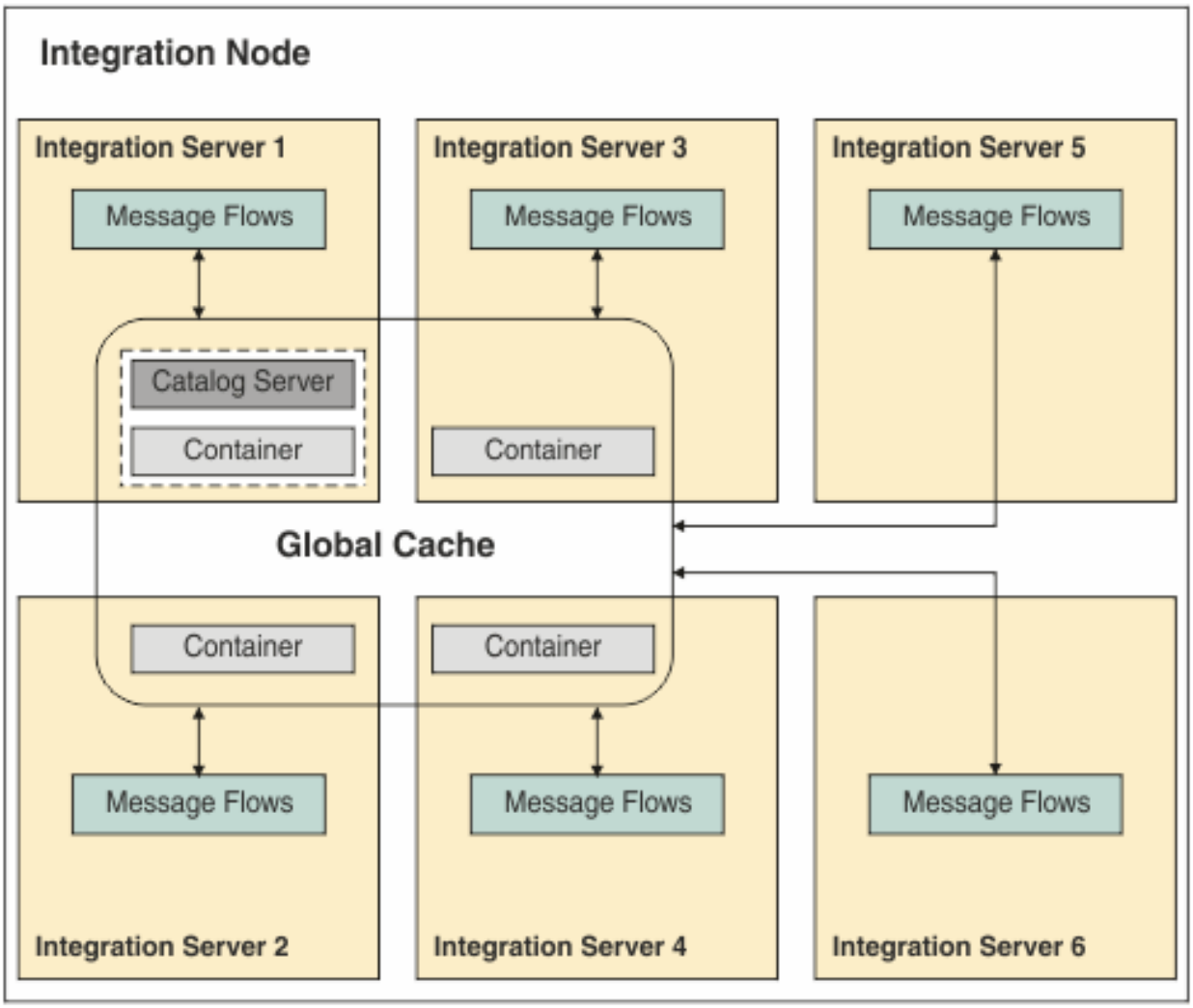
Requirement	Embedded global cache	External WXS grid
Configure your preferred locking strategy, eviction strategy, and copy mode		✓
Use plug-ins (for example, eviction or loading)		✓
Access other configuration options by using WebSphere eXtreme Scale `objectgrid.xml` and `deployment.xml` files		✓
Configure the grid name, zones, quorum, and heartbeat settings on the components		✓
Enterprise grid capability, which can be used by multiple applications		✓
Administer and monitor your grid outside of IBM App Connect Enterprise		✓
Multi-master replication to synchronize grids in different data centers		✓

Data caching terminology

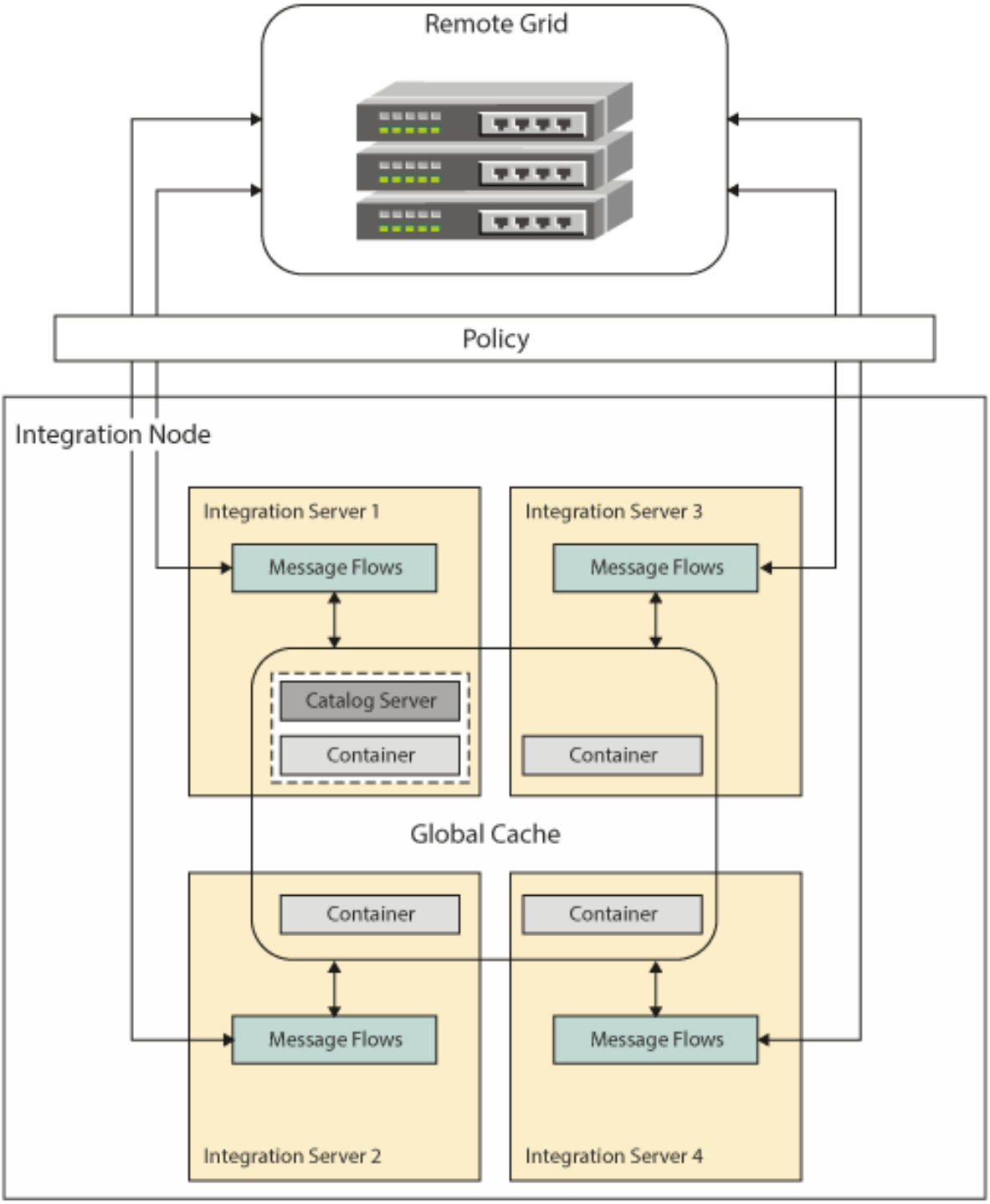
The global cache is embedded in IBM App Connect Enterprise. You can also connect to an external WebSphere eXtreme Scale grid.

The embedded cache can be used by message flows running in any integration server, and you can set properties explicitly for each integration server. The examples in this section show integration servers that are managed by an integration node; however, the global cache can also be used by independent integration servers, which are not managed by an integration node.

The following diagram shows the embedded global cache in an integration node that contains six integration servers. Four integration servers host components for the global cache, but message flows in all six integration servers can use the cache.



The following diagram shows how IBM App Connect Enterprise can connect to both an embedded cache and an external WebSphere eXtreme Scale grid. A policy is used to connect to the external grid.



The following components are involved in the global cache:

Catalog servers

The catalog server is a component that is embedded in an integration server, and it controls the placement of data and monitors the health of containers. You must have at least one catalog server in your global cache.

To avoid losing cache data when a catalog server is lost, you can specify more than one catalog server. If the cache is shared by two integration servers, each of which hosts a catalog server, if one catalog server fails, the remaining catalog server can still be accessed. Having more than one catalog server can affect startup time until the cache is available.

When you are using multiple catalog servers, you can improve performance by taking the following steps:

- Provide other integration servers that host container servers only, rather than having only integration servers that host both catalog and container servers.
- Start and stop integration servers in sequence, rather than using the **mqsistart** or **mqsistop** commands to start or stop all integration servers at once. For example, start the integration servers that host catalog servers before you start the integration servers that host only container servers.

Container servers
A container server is a component that is embedded in the integration server that holds a subset of the cache data. Between them, all container servers in the global cache host all of the cache data at least once. If more than one container exists, the default cache policy ensures that all data is replicated at least once. In this way, the global cache can cope with the loss of container servers without losing data.

You can host more than one container server in a multi-instance integration node. If the active instance of the multi-instance integration node fails, the global cache switches to the container server in the standby instance.

Catalog domain name
When you are using a global cache that spans multiple integration nodes, ensure that all WebSphere eXtreme Scale servers that are clustered in one embedded grid use the same domain name. Only servers with the same domain name can participate in the same grid. WebSphere eXtreme Scale clients use the domain name to identify and distinguish between embedded grids.

Servers in different domains cannot collaborate in the same grid.

Grids
WebSphere eXtreme Scale provides a scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages data across multiple servers. The catalog servers and container servers for the IBM App Connect Enterprise global cache collaborate to act as a WebSphere eXtreme Scale grid. For more information about grids, see WebSphere Extreme Scale product documentation online.

Maps
Data is stored in maps. A map is a data structure that maps keys to values. One map is the default map, but the global cache can have several maps.

The cache uses WebSphere eXtreme Scale dynamic maps. Any map name is allowed, apart from names that begin with SYSTEM.BROKER, which is reserved for use by the integration node. The default map is named SYSTEM.BROKER.DEFAULTMAP; you can use or clear this map.

ObjectGrid® file
An ObjectGrid XML file is used to configure the WebSphere eXtreme Scale client. You can use this file to override WebSphere eXtreme Scale properties. For more information about configuring clients, see WebSphere Extreme Scale product documentation online.

You can configure WebSphere eXtreme Scale options by using the WXSServer policy. For more information, see “Connecting to a WebSphere eXtreme Scale grid” on page 309.

You can use resource statistics and activity trace to monitor the status of the global cache and external grid, and to diagnose problems. You can also administer the embedded global cache by using the **mqsicacheadmin** command.

Configuring the embedded global cache

Configure the embedded global cache by setting properties in the `server.conf.yaml` configuration files.

Before you begin

- For concept information, see "Data caching overview" on page 272.
- When you are using the global cache, increase the JVM heap size for integration servers that are hosting cache components. For more information, see the *Planning environment capacity* section of the WebSphere Extreme Scale product documentation online.

About this task

To use the global cache, set properties in the `server.conf.yaml` configuration files for the integration servers that will share the global cache. By default, the global cache is turned off.

You can configure the global cache by directly setting the properties in the **GlobalCache** section of the `server.conf.yaml` files for your integration servers. Alternatively, you can copy and paste the content of the global cache section to replace the existing content of the **GlobalCache** section of your `server.conf.yaml` files, and then modify as required. The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` files. The sample files show how to configure the global cache for different scenarios, from a basic configuration using a single integration server that is hosting both a catalog server and a container server, to more complex configurations involving multiple integration servers.

The following sample configuration files are provided with IBM App Connect Enterprise, in the *install_directory*`\server\samples\globalcache` directory. These files contain examples of various global cache configurations that you might require:

- **basic_1_catalog_1_container**

  This scenario has one integration server, which hosts both a catalog server and a container server. It is recommended for development purposes.

- **basic_1_catalog_4_containers**

  This scenario has four integration servers: one integration server hosts both a catalog server and a container server, and the other three integration servers host a container server each.

- **basic_2_catalogs_4_containers**

  This scenario has four integration servers: two integration servers host both a catalog server and a container server, and the other two integration servers host a container server each.

- **ha_multi_instance**

  This scenario has four integration servers: two integration servers host both a catalog server and a container server, and the other two integration servers host a container server each and must be used as part of a multi-instance integration node. In this scenario, a multi-instance integration node cannot be used to host an integration server with a catalog server.

Each of these sample files contains the global cache section of a `server.conf.yaml` configuration file. You can copy and paste the content of the global cache section to replace the existing content of the **GlobalCache** section of your `server.conf.yaml` files, and then modify as required.

Depending on the type of data you store in your cache, you can configure your embedded global cache to use different locking strategies, and you can configure the use of replica shards for read access. For more information, see "Optimizing the embedded global cache for use with different types of cache data" on page 305.

If you stop the integration server that contains the catalog server, the cache becomes unavailable. If you restart the integration server that hosts the catalog server, it can no longer communicate with the container servers in other integration servers. Although these container servers are still running, they are

no longer part of the cache, and your data is lost. Therefore, you must also restart the integration servers that host the container servers.

Procedure

Configure the global cache by completing the following steps:

1. Open the configuration file for your integration server (`server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file. For more information about configuring an integration server, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
2. Set the properties in the **`GlobalCache`** section of the `server.conf.yaml` configuration files, either directly or by copying and pasting the content of sample global cache configuration files to replace the existing content of the **`GlobalCache`** section.

   The `server.conf.yaml` and sample configuration files contain information about the valid properties.

   For information about configuring the global cache by using the sample configuration files, see the following topics:

   - "Global cache sample: Configuring one catalog and one container" on page 286
   - "Global cache sample: Configuring one catalog and four containers" on page 288
   - "Global cache sample: Configuring two catalogs and four containers" on page 294
   - "Global cache sample: Configuring for high availability (multi-instance)" on page 300
3. When you have modified and saved a `server.conf.yaml` file, restart the integration server for the changes to take effect.

   For information about how to start an integration server, see "Starting an integration server" on page 250.

Global cache sample: Configuring one catalog and one container

Use this sample to create a basic global cache configuration, in which a single integration server hosts both a catalog server and a container server.

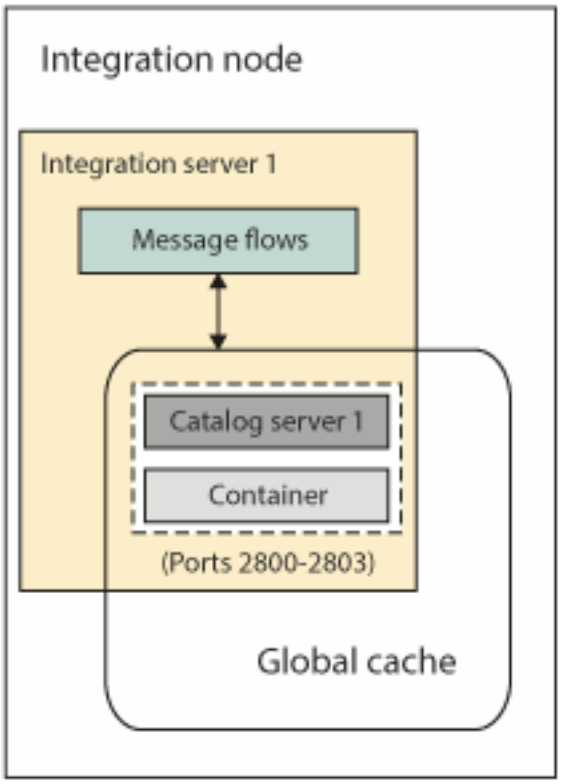
Before you begin

- Read the task topic "Configuring the embedded global cache" on page 285.
- For concept information, see "Data caching overview" on page 272.
- When you are using the global cache, increase the JVM heap size for integration servers that are hosting cache components. For more information, see the *Planning environment capacity* section of the WebSphere Extreme Scale product documentation online.

About this task

You can copy and paste the content of the global cache section of configuration files to replace the existing content of the **`GlobalCache`** section of your `server.conf.yaml` files and then modify them as required. The **`GlobalCache`** section is a sub-section of the **`ResourceManagers`** section in the `server.conf.yaml` files.

This sample shows how to configure the global cache for a basic scenario, using a single integration server that is hosting both a catalog server and a container server, as shown in the following diagram:



For information about other samples, which show how to configure global cache for different scenarios, see the following topics:

- "Global cache sample: Configuring one catalog and four containers" on page 288
- "Global cache sample: Configuring two catalogs and four containers" on page 294
- "Global cache sample: Configuring for high availability (multi-instance)" on page 300

Procedure

Complete the following steps to create a basic global cache configuration, in which a single integration server hosts both a catalog server and a container server:

1. Create an integration server called `integrationServer1`.
2. Open the configuration file for your integration server (`integrationServer1\server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
3. Copy and paste the content of the **GlobalCache** section in the supplied `server.conf.yaml` sample file (in the *`install_directory`*`\server\samples\globalcache\basic_1_catalog_1_container` folder) to replace the existing content of the **GlobalCache** section of your `server.conf.yaml` file.

   The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` file.

```
# Integration server configuration file for use as global cache container
# Provides a "basic" configuration with a single integration server acting as both a catalog
 server and a container server
#
# General notes:
#  - Integration server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
```

```
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                    # Set to true to enable Global Cache
 functionality
                                                     # When using Global Cache it is advisable
 to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                     # the number of live objects in the heap,
 complexity of live objects in the heap and number of available cores.
                                                     # see https://www.ibm.com/support/
knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer1'              # The name of this cache server component
 (a cache server component can be a catalog and/or a container); it must be unique in your
 global cache system
    catalogServiceEndPoints: 'localhost:2800'        # Comma-separated list of hostnames and
 ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'           # Name of the shared global cache domain;
 this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints: 'MyCatalogServer1:localhost:2803:2801'
                                                     # Comma-separated list of catalog server
 connection details in the format 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                     # If this is a catalog server,
 cacheServerName should match the value above, and if not, it will be the value used on the
 integration server hosting it
                                                     # The list should be in the same order
 for all catalog and container servers which are interacting together in the same domain
    enableCatalogService: true                       # Set to true to launch a catalog service
 cache server component in this integration server
    enableContainerService: true                     # Set to true to launch a container
 service cache server component in this integration server
    enableJMX: true                                  # Allow admin access to this container
 service via JMX
    listenerHost: 'localhost'                        # Comma-separated list of hostnames for
 this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2800                               # Port number this cache server listens
 on; it must be unique on this machine
                                                     # Four consecutive ports are assigned,
 e.g. 2800 for catalogCacheServerListenerPort, 2801 for clientPort, 2802 for JMXServicePort,
 2803 for HAPort

    #deploymentPolicyCustomFile: ''                  # Override the deployment policy  file
 (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                        # Override the ObjectGrid file (default
 is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                           # Set a trace level for the cache server
 components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                      # Set to true to enable SSL for any
 client connections to the cache servers
    #sslAlias: ''                                    # SSL Alias to use for the cache server
 component
    #sslProtocol: ''                                 # SSL Protocol to use for SSL connections
 eg. default is "TLSv1.2"
```

4. Restart the integration server for the changes to take effect.

   When it is restarted, `integrationServer1` will be both a catalog server and a container server.

What to do next

If you have only one catalog server and you stop it, the cache becomes unavailable and your data is lost. However, if you have multiple catalog servers, one of them can be stopped and restarted without losing data; your data is lost only if all catalog servers are stopped.

Global cache sample: Configuring one catalog and four containers

Use this sample to create a global cache configuration involving four integration servers, one of which hosts both a catalog server and a container server, while the other three integration servers host a container server each.

Before you begin

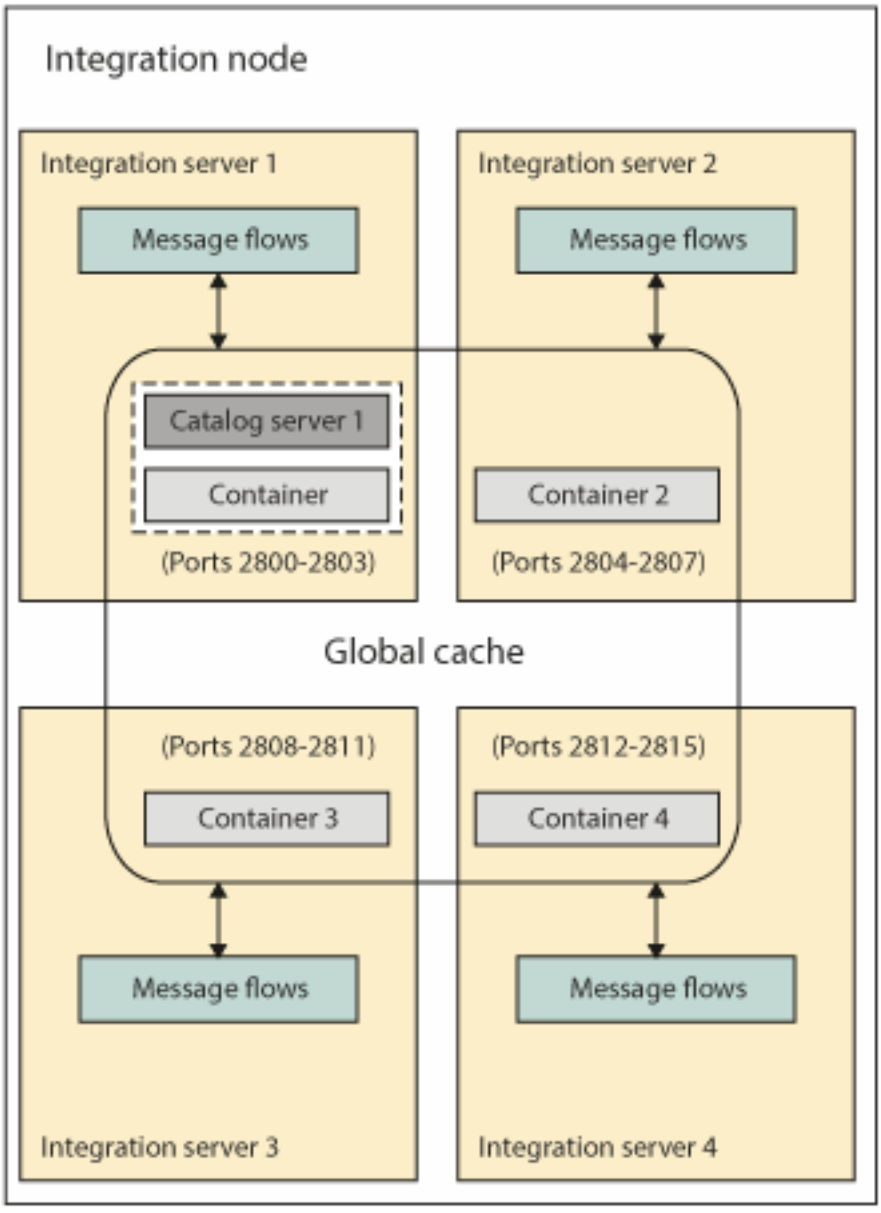
- Read the task topic “Configuring the embedded global cache” on page 285.
- For concept information, see “Data caching overview” on page 272.

- When you are using the global cache, increase the JVM heap size for integration servers that are hosting cache components. For more information, see the *Planning environment capacity* section of the WebSphere Extreme Scale product documentation online.

About this task

You can copy and paste the content of the global cache section of configuration files to replace the existing content of the **GlobalCache** section of your `server.conf.yaml` files and then modify them as required. The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` files.

This sample shows how to configure a global cache to be used by four integration servers: one integration server hosts both a catalog server and a container server, and the other three integration servers host a container server each, as shown in the following diagram:



This scenario is equivalent to the default cache policy in previous versions of IBM Integration Bus and WebSphere Message Broker. It is also equivalent to the "policy_two_brokers" cache policy XML file, which used one catalog server.

For information about other samples, which show how to configure global cache for different scenarios, see the following topics:

- “Global cache sample: Configuring one catalog and one container” on page 286
- “Global cache sample: Configuring two catalogs and four containers” on page 294
- “Global cache sample: Configuring for high availability (multi-instance)” on page 300

Procedure

Complete the following steps to create a global cache configuration involving four integration servers, one of which hosts both a catalog server and a container server, and the other three host a container server each:

1. Create four integration servers called `integrationServer1`, `integrationServer2`, `integrationServer3`, and `integrationServer4`.
2. Open the configuration files for your integration servers (`integrationServer<n>\server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
3. Copy and paste the content of the **GlobalCache** section in the supplied `server.conf.yaml` sample files (in the *`install_directory`*`\server\samples\globalcache\basic_1_catalog_4_containers` folder) to replace the existing content of the **GlobalCache** section of your corresponding `integrationServer<n>\server.conf.yaml` files.

   The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` files.

   integrationServer1.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "default" configuration in previous versions of IBM
 Integration Bus and Websphere Message Broker
# Uses four integration servers where one is both a catalog server and a container server,
 and the other three are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                    # Set to true to enable Global Cache
 functionality
                                                     # When using Global Cache it is advisable
 to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                     # the number of live objects in the heap,
 complexity of live objects in the heap and number of available cores.
                                                     # see https://www.ibm.com/support/
knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer1'              # The name of this cache server component
 (a cache server component can be a catalog and/or a container); it must be unique in your
 global cache system
    catalogServiceEndPoints: 'localhost:2800'        # Comma-separated list of hostnames and
 ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'           # Name of the shared global cache domain;
 this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints: 'MyCatalogServer1:localhost:2803:2801'
                                                     # Comma-separated list of catalog server
 connection details in the format 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                     # If this is a catalog server,
 cacheServerName should match the value above, and if not, it will be the value used on the
 integration server hosting it
                                                     # The list should be in the same order
 for all catalog and container servers which are interacting together in the same domain
    enableCatalogService: true                       # Set to true to launch a catalog service
 cache server component in this integration server
```

```
    enableContainerService: true                    # Set to true to launch a container
 service cache server component in this integration server
    enableJMX: true                                 # Allow admin access to this container
 service via JMX
    listenerHost: 'localhost'                       # Comma-separated list of hostnames for
 this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2800                              # Port number this cache server listens
 on; it must be unique on this machine
                                                    # Four consecutive ports are assigned,
 e.g. 2800 for catalogCacheServerListenerPort, 2801 for clientPort, 2802 for JMXServicePort,
 2803 for HAPort

    #deploymentPolicyCustomFile: ''                 # Override the deployment policy file
 (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                       # Override the ObjectGrid file (default
 is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                          # Set a trace level for the cache server
 components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                     # Set to true to enable SSL for any
 client connections to the cache servers
    #sslAlias: ''                                   # SSL Alias to use for the cache server
 component
    #sslProtocol: ''                                # SSL Protocol to use for SSL connections
 eg. default is "TLSv1.2"
```

integrationServer2.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "default" configuration in previous versions of IBM
 Integration Bus and Websphere Message Broker
# Uses four integration servers where one is both a catalog server and a container server,
 and the other three are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                   # Set to true to enable Global Cache
 functionality
                                                    # When using Global Cache it is advisable
 to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                    # the number of live objects in the heap,
 complexity of live objects in the heap and number of available cores.
                                                    # see https://www.ibm.com/support/
knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer2'           # The name of this cache server component
 (a cache server component can be a catalog and/or a container); it must be unique in your
 global cache system
    catalogServiceEndPoints: 'localhost:2800'       # Comma-separated list of hostnames and
 ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'          # Name of the shared global cache domain;
 this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints: 'MyCatalogServer1:localhost:2803:2801'
                                                    # Comma-separated list of catalog server
 connection details in the format 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                    # If this is a catalog server,
 cacheServerName should match the value above, and if not, it will be the value used on the
 integration server hosting it
                                                    # The list should be in the same order
 for all catalog and container servers which are interacting together in the same domain
    enableCatalogService: false                     # Set to true to launch a catalog service
 cache server component in this integration server
    enableContainerService: true                    # Set to true to launch a container
 service cache server component in this integration server
    enableJMX: true                                 # Allow admin access to this container
 service via JMX
    listenerHost: 'localhost'                       # Comma-separated list of hostnames for
 this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2804                              # Port number this cache server listens
 on; it must be unique on this machine
                                                    # Four consecutive ports are assigned,
 e.g. 2804 for catalogCacheServerListenerPort, 2805 for clientPort, 2806 for JMXServicePort,
 2807 for HAPort

    #deploymentPolicyCustomFile: ''                 # Override the deployment policy file
 (default is <install directory>/server/cachesupport/config/deployment.xml)
```

```
    #objectGridCustomFile: ''                       # Override the ObjectGrid file (default
 is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                          # Set a trace level for the cache server
 components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                     # Set to true to enable SSL for any
 client connections to the cache servers
    #sslAlias: ''                                   # SSL Alias to use for the cache server
 component
    #sslProtocol: ''                                # SSL Protocol to use for SSL connections
 eg. default is "TLSv1.2"
```

integrationServer3.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "default" configuration in previous versions of IBM
 Integration Bus and Websphere Message Broker
# Uses four integration servers where one is both a catalog server and a container server,
 and the other three are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                   # Set to true to enable Global Cache
 functionality
                                                    # When using Global Cache it is advisable
 to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                    # the number of live objects in the heap,
 complexity of live objects in the heap and number of available cores.
                                                    # see https://www.ibm.com/support/
knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer3'           # The name of this cache server component
 (a cache server component can be a catalog and/or a container); it must be unique in your
 global cache system
    catalogServiceEndPoints: 'localhost:2800'       # Comma-separated list of hostnames and
 ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'          # Name of the shared global cache domain;
 this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints: 'MyCatalogServer1:localhost:2803:2801'
                                                    # Comma-separated list of catalog server
 connection details in the format 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                    # If this is a catalog server,
 cacheServerName should match the value above, and if not, it will be the value used on the
 integration server hosting it
                                                    # The list should be in the same order
 for all catalog and container servers which are interacting together in the same domain
    enableCatalogService: false                     # Set to true to launch a catalog service
 cache server component in this integration server
    enableContainerService: true                    # Set to true to launch a container
 service cache server component in this integration server
    enableJMX: true                                 # Allow admin access to this container
 service via JMX
    listenerHost: 'localhost'                       # Comma-separated list of hostnames for
 this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2808                              # Port number this cache server listens
 on; it must be unique on this machine
                                                    # Four consecutive ports are assigned,
 e.g. 2808 for catalogCacheServerListenerPort, 2809 for clientPort, 2810 for JMXServicePort,
 2811 for HAPort

    #deploymentPolicyCustomFile: ''                 # Override the deployment policy file
 (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                       # Override the ObjectGrid file (default
 is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                          # Set a trace level for the cache server
 components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                     # Set to true to enable SSL for any
 client connections to the cache servers
    #sslAlias: ''                                   # SSL Alias to use for the cache server
 component
```

```
    #sslProtocol: ''                                # SSL Protocol to use for SSL connections
 eg. default is "TLSv1.2"
```

integrationServer4.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "default" configuration in previous versions of IBM
 Integration Bus and Websphere Message Broker
# Uses four integration servers where one is both a catalog server and a container server,
 and the other three are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                   # Set to true to enable Global Cache
 functionality
                                                    # When using Global Cache it is advisable
 to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                    # the number of live objects in the heap,
 complexity of live objects in the heap and number of available cores.
                                                    # see https://www.ibm.com/support/
knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer4'           # The name of this cache server component
 (a cache server component can be a catalog and/or a container); it must be unique in your
 global cache system
    catalogServiceEndPoints: 'localhost:2800'       # Comma-separated list of hostnames and
 ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'          # Name of the shared global cache domain;
 this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints: 'MyCatalogServer1:localhost:2803:2801'
                                                    # Comma-separated list of catalog server
 connection details in the format 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                    # If this is a catalog server,
 cacheServerName should match the value above, and if not, it will be the value used on the
 integration server hosting it
                                                    # The list should be in the same order
 for all catalog and container servers which are interacting together in the same domain
    enableCatalogService: false                     # Set to true to launch a catalog service
 cache server component in this integration server
    enableContainerService: true                    # Set to true to launch a container
 service cache server component in this integration server
    enableJMX: true                                 # Allow admin access to this container
 service via JMX
    listenerHost: 'localhost'                       # Comma-separated list of hostnames for
 this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2812                              # Port number this cache server listens
 on; it must be unique on this machine
                                                    # Four consecutive ports are assigned,
 e.g. 2812 for catalogCacheServerListenerPort, 2813 for clientPort, 2814 for JMXServicePort,
 2815 for HAPort

    #deploymentPolicyCustomFile: ''                 # Override the deployment policy file
 (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                       # Override the ObjectGrid file (default
 is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                          # Set a trace level for the cache server
 components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                     # Set to true to enable SSL for any
 client connections to the cache servers
    #sslAlias: ''                                   # SSL Alias to use for the cache server
 component
    #sslProtocol: ''                                # SSL Protocol to use for SSL connections
 eg. default is "TLSv1.2"
```

4. Restart the integration servers for the changes to take effect.

   When they are restarted, `integrationServer1` will be both a catalog server and a container server, and `integrationServer2`, `integrationServer3`, and `integrationServer4` will be container servers.

What to do next

If you have only one catalog server and you stop it, the cache becomes unavailable and your data is lost. However, if you have multiple catalog servers, one of them can be stopped and restarted without losing data; your data is lost only if all catalog servers are stopped.

Global cache sample: Configuring two catalogs and four containers

Use this sample to create a global cache configuration involving four integration servers, two of which host both a catalog server and a container server, while the other two integration servers host a container server each.

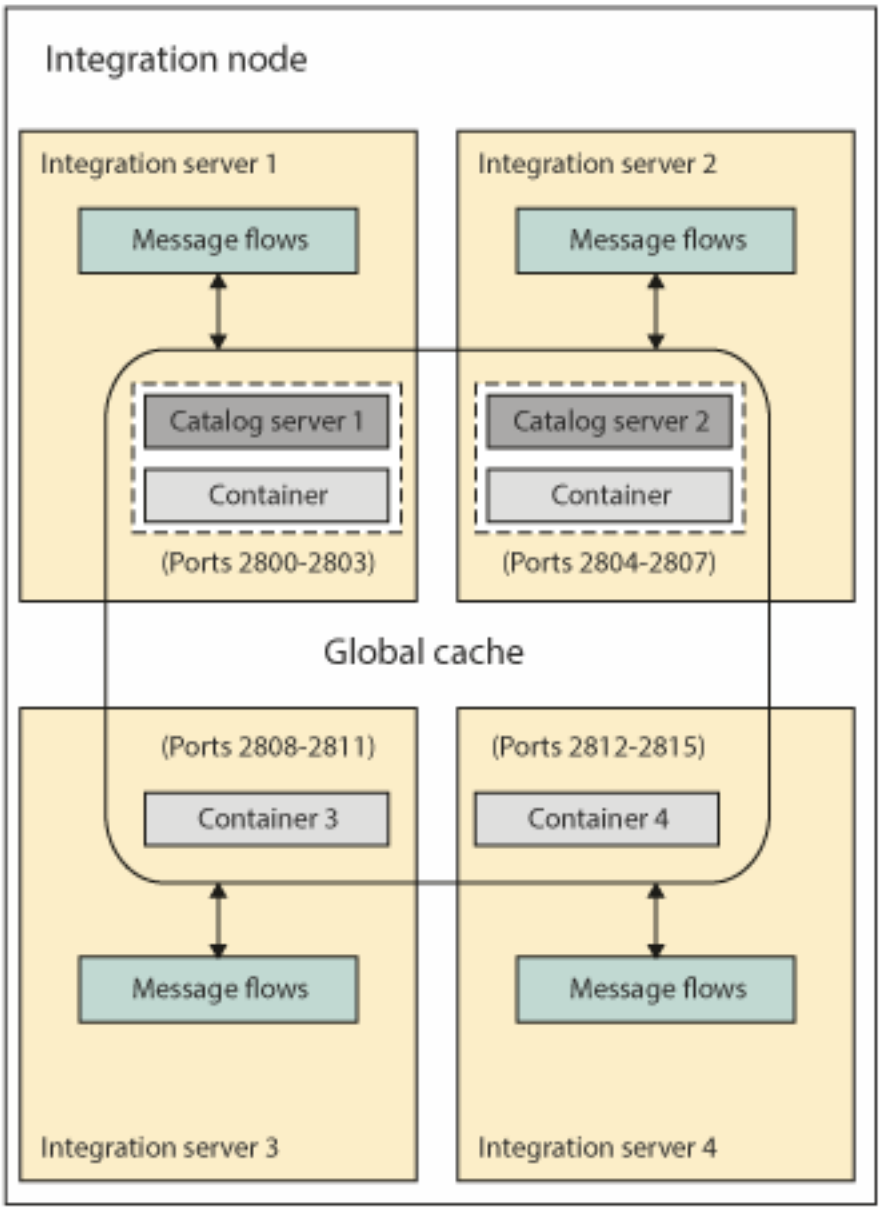
Before you begin

- Read the task topic "Configuring the embedded global cache" on page 285.
- For concept information, see "Data caching overview" on page 272.
- When you are using the global cache, increase the JVM heap size for integration servers that are hosting cache components. For more information, see the *Planning environment capacity* section of the WebSphere Extreme Scale product documentation online.

About this task

You can copy and paste the content of the global cache section of configuration files to replace the existing content of the **`GlobalCache`** section of your `server.conf.yaml` files and then modify them as required. The **`GlobalCache`** section is a sub-section of the **`ResourceManagers`** section in the `server.conf.yaml` files.

This sample shows how to configure a global cache to be used by four integration servers: two integration servers host both a catalog server and a container server, and the other two integration servers host a container server each, as shown in the following diagram:



This scenario is equivalent to the "policy_one_broker_ha" and "policy_two_brokers_ha" cache policy XML files in previous versions of IBM Integration Bus and WebSphere Message Broker. It covers a similar high-availability configuration as the two XML files, by making use of two catalog servers.

For information about other samples, which show how to configure global cache for different scenarios, see the following topics:

- “Global cache sample: Configuring one catalog and one container” on page 286
- “Global cache sample: Configuring one catalog and four containers” on page 288
- “Global cache sample: Configuring for high availability (multi-instance)” on page 300

Procedure

Complete the following steps to create a global cache configuration involving four integration servers, two of which host both a catalog server and a container server, and the other two host a container server each:

1. Create four integration servers called `integrationServer1`, `integrationServer2`, `integrationServer3`, and `integrationServer4`.

2. Open the configuration files for your integration servers (`integrationServer<n>\server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

3. Copy and paste the content of the **GlobalCache** section in the supplied `server.conf.yaml` sample files (in the *install_directory*`\server\samples\globalcache\basic_2_catalogs_4_containers` folder) to replace the existing content of the **GlobalCache** section of your corresponding `integrationServer<n>\server.conf.yaml` files.

   The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` files.

   integrationServer1.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "One Broker HA" and Two Brokers HA" Cache Policy XML
 configurations in previous versions of IBM Integration Bus and Websphere Message Broker
# Uses four integration servers where two are both a catalog server and a container server,
 and the other two are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                                 # Set to true to enable
 Global Cache functionality
                                                                  # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                                  # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                                  # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer1'                           # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'localhost:2800,localhost:2804'    # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                        # Name of the shared global
 cache domain; this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:localhost:2803:2801,MyCatalogServer2:localhost:2807:2805'
                                                                  #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                                  # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                                  # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
    enableCatalogService: true                                    # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                                  # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                               # Allow admin access to this
 container service via JMX
    listenerHost: 'localhost'                                     # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2800                                            # Port number this cache
 server listens on; it must be unique on this machine
                                                                  # Four consecutive ports
 are assigned, e.g. 2800 for catalogCacheServerListenerPort, 2801 for clientPort, 2802 for
 JMXServicePort, 2803 for HAPort

    #deploymentPolicyCustomFile: ''                               # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                                     # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
```

```
    #overrideTraceSpec: ''                                  # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                             # Set to true to enable SSL
 for any client connections to the cache servers
    #sslAlias: ''                                           # SSL Alias to use for the
 cache server component
    #sslProtocol: ''                                        # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

integrationServer2.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "One Broker HA" and Two Brokers HA" Cache Policy XML
 configurations in previous versions of IBM Integration Bus and Websphere Message Broker
# Uses four integration servers where two are both a catalog server and a container server,
 and the other two are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                           # Set to true to enable
 Global Cache functionality
                                                            # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                            # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                            # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer2'                     # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'localhost:2800,localhost:2804'    # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                  # Name of the shared global
 cache domain; this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:localhost:2803:2801,MyCatalogServer2:localhost:2807:2805'
                                                            #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                            # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                            # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
    enableCatalogService: true                              # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                            # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                         # Allow admin access to this
 container service via JMX
    listenerHost: 'localhost'                               # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2804                                      # Port number this cache
 server listens on; it must be unique on this machine
                                                            # Four consecutive ports
 are assigned, e.g. 2804 for catalogCacheServerListenerPort, 2805 for clientPort, 2806 for
 JMXServicePort, 2807 for HAPort

    #deploymentPolicyCustomFile: ''                         # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                               # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                                  # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                             # Set to true to enable SSL
 for any client connections to the cache servers
    #sslAlias: ''                                           # SSL Alias to use for the
 cache server component
    #sslProtocol: ''                                        # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

integrationServer3.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "One Broker HA" and Two Brokers HA" Cache Policy XML
 configurations in previous versions of IBM Integration Bus and Websphere Message Broker
# Uses four integration servers where two are both a catalog server and a container server,
 and the other two are container servers
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                           # Set to true to enable
 Global Cache functionality
                                                            # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                            # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                            # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer3'                   # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'localhost:2800,localhost:2804'    # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                  # Name of the shared global
 cache domain; this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:localhost:2803:2801,MyCatalogServer2:localhost:2807:2805'
                                                            #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                            # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                            # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
    enableCatalogService: false                             # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                            # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                         # Allow admin access to this
 container service via JMX
    listenerHost: 'localhost'                               # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2808                                      # Port number this cache
 server listens on; it must be unique on this machine
                                                            # Four consecutive ports
 are assigned, e.g. 2808 for catalogCacheServerListenerPort, 2809 for clientPort, 2810 for
 JMXServicePort, 2811 for HAPort

    #deploymentPolicyCustomFile: ''                         # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                               # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                                  # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                             # Set to true to enable SSL
 for any client connections to the cache servers
    #sslAlias: ''                                           # SSL Alias to use for the
 cache server component
    #sslProtocol: ''                                        # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

integrationServer4.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "One Broker HA" and Two Brokers HA" Cache Policy XML
 configurations in previous versions of IBM Integration Bus and Websphere Message Broker
# Uses four integration servers where two are both a catalog server and a container server,
 and the other two are container servers
#
```

```
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                           # Set to true to enable
 Global Cache functionality
                                                            # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                            # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                            # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer4'                   # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'localhost:2800,localhost:2804'    # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                  # Name of the shared global
 cache domain; this value should be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:localhost:2803:2801,MyCatalogServer2:localhost:2807:2805'
                                                            #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                            # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                            # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
    enableCatalogService: false                             # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                            # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                         # Allow admin access to this
 container service via JMX
    listenerHost: 'localhost'                               # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2812                                      # Port number this cache
 server listens on; it must be unique on this machine
                                                            # Four consecutive ports
 are assigned, e.g. 2812 for catalogCacheServerListenerPort, 2813 for clientPort, 2814 for
 JMXServicePort, 2815 for HAPort

    #deploymentPolicyCustomFile: ''                         # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                               # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                                  # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: false                             # Set to true to enable SSL
 for any client connections to the cache servers
    #sslAlias: ''                                           # SSL Alias to use for the
 cache server component
    #sslProtocol: ''                                        # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

4. Restart the integration servers for the changes to take effect.

   When they are restarted, `integrationServer1` and `integrationServer2` will be both catalog servers and container servers, while `integrationServer3` and `integrationServer4` will be container servers.

What to do next

If you have only one catalog server and you stop it, the cache becomes unavailable and your data is lost. However, if you have multiple catalog servers, one of them can be stopped and restarted without losing data; your data is lost only if all catalog servers are stopped.

Global cache sample: Configuring for high availability (multi-instance)

Use this sample to create a global cache configuration involving three integration servers, two of which host both a catalog server and a container server. The other integration server hosts only a container server, and must be used as part of a multi-instance integration node.

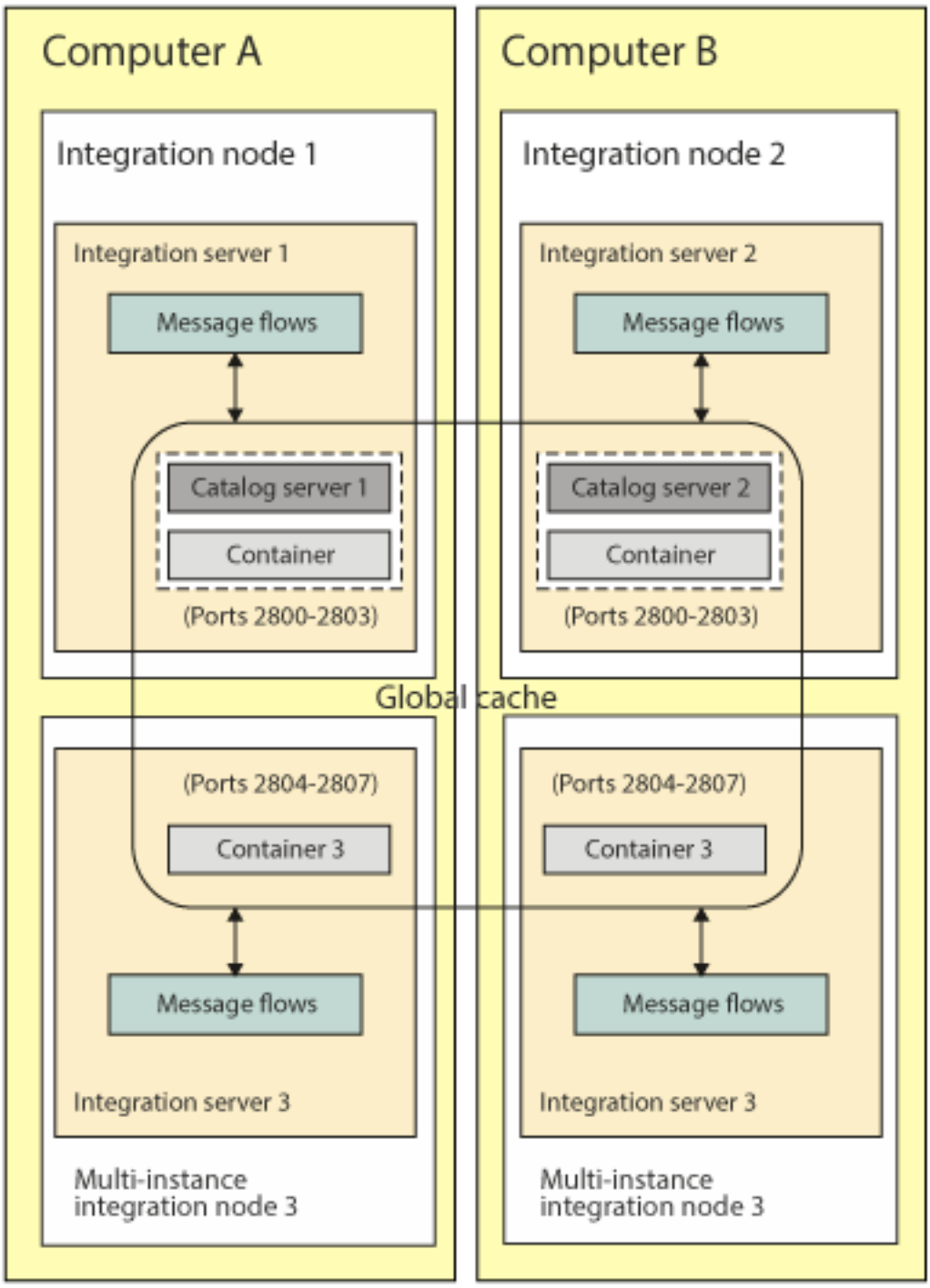
Before you begin

- Read the task topic "Configuring the embedded global cache" on page 285.
- For concept information, see "Data caching overview" on page 272.
- When you are using the global cache, increase the JVM heap size for integration servers that are hosting cache components. For more information, see the *Planning environment capacity* section of the WebSphere Extreme Scale product documentation online.

About this task

You can copy and paste the content of the global cache section of configuration files to replace the existing content of the **`GlobalCache`** section of your `server.conf.yaml` files and then modify them as required. The **`GlobalCache`** section is a sub-section of the **`ResourceManagers`** section in the `server.conf.yaml` files.

This sample shows how to configure a global cache to be used by three integration servers: two integration servers host both a catalog server and a container server, while the other integration server hosts only a container server, and must be used as part of a multi-instance integration node. In this scenario, a multi-instance integration node cannot be used to host an integration server with a catalog server.



This scenario is equivalent to the `policy_multi_instance` cache policy XML file in previous versions of IBM Integration Bus and WebSphere Message Broker. It covers a similar multi-instance integration node configuration as the XML file.

For information about other samples, which show how to configure global cache for different scenarios, see the following topics:

- "Global cache sample: Configuring one catalog and one container" on page 286
- "Global cache sample: Configuring one catalog and four containers" on page 288
- "Global cache sample: Configuring two catalogs and four containers" on page 294

Procedure

Complete the following steps to create a global cache configuration involving three integration servers, two of which host both a catalog server and a container server, while the other integration server hosts only a container server, and must be used as part of a multi-instance integration node:

1. Create two integration servers called `integrationServer1` and `integrationServer2`.
2. Open the configuration files for your integration servers (`integrationServer1\server.conf.yaml` and `integrationServer2\server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
3. Copy and paste the content of the **GlobalCache** section from the supplied `integrationServer1.server.conf.yaml` and `integrationServer2.server.conf.yaml` sample files in the *install_directory*`\server\samples\globalcache\ha_multi_instance` folder, to replace the existing content of the **GlobalCache** sections of your corresponding `integrationServer1\server.conf.yaml` and `integrationServer2\server.conf.yaml` files.

   The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` file.

   integrationServer1.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "Muti-Instance" Cache Policy XML configuration in previous
 versions of IBM Integration Bus and Websphere Message Broker
# Uses two integration servers (1 and 2) that are both a catalog server and a container
 server
# It also uses an integration server (3) which is a container server; this integration server
 must be part of a multi-instance integration node
# A multi-instance integration node cannot be used to host an integration server with a
 catalog server
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                                  # Set to true to enable
 Global Cache functionality
                                                                   # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                                   # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                                   # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer1'                            # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'server1.mycompany.com:2800,server2.mycompany.com:2800'
                                                                   # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                         # Name of the shared global
 cache domain; this value will be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:server1.mycompany.com:2803:2801,MyCatalogServer2:server2.mycompany.com:2803:2801'
                                                                   #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                                   # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                                   # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
```

```
    enableCatalogService: true                                  # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                                # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                             # Allow admin access to this
 container service via JMX
    listenerHost: 'server1.mycompany.com'                       # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
    listenerPort: 2800                                          # Port number this cache
 server listens on; it must be unique on this machine
                                                                # Four consecutive ports
 are assigned, e.g. 2800 for catalogCacheServerListenerPort, 2801 for clientPort, 2802 for
 JMXServicePort, 2803 for HAPort

    #deploymentPolicyCustomFile: ''                             # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
    #objectGridCustomFile: ''                                   # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
    #overrideTraceSpec: ''                                      # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
    #clientsDefaultToSSL: ''                                    # Set to true to enable SSL
 for any client connections to the cache servers
    #sslAlias: ''                                               # SSL Alias to use for the
 cache server components
    #sslProtocol: ''                                            # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

integrationServer2.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "Muti-Instance" Cache Policy XML configuration in previous
 versions of IBM Integration Bus and Websphere Message Broker
# Uses two integration servers (1 and 2) that are both a catalog server and a container
 server
# It also uses an integration server (3) which is a container server; this integration server
 must be part of a multi-instance integration node
# A multi-instance integration node cannot be used to host an integration server with a
 catalog server
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                               # Set to true to enable
 Global Cache functionality
                                                                # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                                # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                                # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyCatalogServer2'                         # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'server1.mycompany.com:2800,server2.mycompany.com:2800'
                                                                # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                      # Name of the shared global
 cache domain; this value will be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:server1.mycompany.com:2803:2801,MyCatalogServer2:server2.mycompany.com:2803:2801'
                                                                #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                                # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
                                                                # The list should be in the
 same order for all catalog and container servers which are interacting together in the same
 domain
    enableCatalogService: true                                  # Set to true to launch a
 catalog service cache server component in this integration server
    enableContainerService: true                                # Set to true to launch a
 container service cache server component in this integration server
    enableJMX: true                                             # Allow admin access to this
 container service via JMX
```

```
   listenerHost: 'server2.mycompany.com'                        # Comma-separated list of
 hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
   listenerPort: 2800                                           # Port number this cache
 server listens on; it must be unique on this machine
                                                                # Four consecutive ports
 are assigned, e.g. 2800 for catalogCacheServerListenerPort, 2801 for clientPort, 2802 for
 JMXServicePort, 2803 for HAPort

   #deploymentPolicyCustomFile: ''                              # Override the deployment
 policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
   #objectGridCustomFile: ''                                    # Override the ObjectGrid
 file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
   #overrideTraceSpec: ''                                       # Set a trace level for the
 cache server components, e.g. ObjectGrid*=event=enabled
   #clientsDefaultToSSL: false                                  # Set to true to enable SSL
 for any client connections to the cache servers
   #sslAlias: ''                                                # SSL Alias to use for the
 cache server component
   #sslProtocol: ''                                             # SSL Protocol to use for SSL
 connections eg. default is "TLSv1.2"
```

4. Create a multi-instance integration node called `MultiInstanceNode` on two servers, as described in "Creating a multi-instance integration node" on page 122.
5. Create an integration server called `integrationServer3` under the multi-instance integration node.
6. Copy and paste the content of the **GlobalCache** section from the supplied `integrationServer3.server.conf.yaml` sample file in the `samples\globalcache\multi_instance` folder, to replace the existing content of the **GlobalCache** section in the *SharedWorkPath*`\components\INODE\servers\IntegrationServer3\server.conf.yaml` file.

   The **GlobalCache** section is a sub-section of the **ResourceManagers** section in the `server.conf.yaml` file.

   integrationServer3.server.conf.yaml:

```
# Integration server configuration file for use as global cache container
# Provides an equivalent to the "Muti-Instance" Cache Policy XML configuration in previous
 versions of IBM Integration Bus and Websphere Message Broker
# Uses two integration servers (1 and 2) that are both a catalog server and a container
 server
# It also uses an integration server (3) which is a container server; this integration server
 must be part of a multi-instance integration node
# A multi-instance integration node cannot be used to host an integration server with a
 catalog server
#
# General notes :
#  - Integration Server will load server.conf.yaml from directory set via --work-dir
#  - To ensure valid YAML avoid any use of TAB characters
#  - File paths may be taken as absolute, or relative to the integration server's work
 directory
#

  GlobalCache:
    cacheOn: true                                               # Set to true to enable
 Global Cache functionality
                                                                # When using Global Cache it
 is advisable to change your jvmMinHeapSize and jvmMaxHeapSize depending on
                                                                # the number of live objects
 in the heap, complexity of live objects in the heap and number of available cores.
                                                                # see https://www.ibm.com/
support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/cxsjvmtune.html
    cacheServerName: 'MyContainerServer3'                       # The name of this cache
 server component (a cache server component can be a catalog and/or a container); it must be
 unique in your global cache system
    catalogServiceEndPoints: 'server1.mycompany.com:2800,server2.mycompany.com:2800'
                                                                # Comma-separated list of
 hostnames and ports for the catalog servers to use, e.g. 'localhost:2800'
    catalogDomainName: 'WMB_MyCacheDomain'                      # Name of the shared global
 cache domain; this value will be shared by all catalog servers in the same domain
    catalogClusterEndPoints:
 'MyCatalogServer1:server1.mycompany.com:2803:2801,MyCatalogServer2:server2.mycompany.com:2803:2801'
                                                                #
 Comma-separated list of catalog server connection details in the format
 'cacheServerName:catalogCacheServerHost:HAPort:clientPort'
                                                                # If this is a catalog
 server, cacheServerName should match the value above, and if not, it will be the value used
 on the integration server hosting it
```

```
                                                        # The list should be in the
same order for all catalog and container servers which are interacting together in the same
domain
   enableCatalogService: false                          # Set to true to launch a
catalog service cache server component in this integration server
   enableContainerService: true                         # Set to true to launch a
container service cache server component in this integration server
   enableJMX: true                                      # Allow admin access to this
container service via JMX
   listenerHost: 'server1.mycompany.com,server2.mycompany.com' # Comma-separated list of
hostnames for this cacheServer component, e.g. 'localhost,myserver.mycompany.com'
   listenerPort: 2804                                   # Port number this cache
server listens on; it must be unique on this machine
                                                        # Four consecutive ports
are assigned, e.g. 2804 for catalogCacheServerListenerPort, 2805 for clientPort, 2806 for
JMXServicePort, 2807 for HAPort

   #deploymentPolicyCustomFile: ''                      # Override the deployment
policy file (default is <install directory>/server/cachesupport/config/deployment.xml)
   #objectGridCustomFile: ''                            # Override the ObjectGrid
file (default is <install directory>/server/cachesupport/config/objectgrid_xio.xml)
   #overrideTraceSpec: ''                               # Set a trace level for the
cache server components, e.g. ObjectGrid*=event=enabled
   #clientsDefaultToSSL: false                          # Set to true to enable SSL
for any client connections to the cache servers
   #sslAlias: ''                                        # SSL Alias to use for the
cache server component
   #sslProtocol: ''                                     # SSL Protocol to use for SSL
connections eg. default is "TLSv1.2"
```

7. Change `server1.mycompany.com` and `server2.mycompany.com` in the files to match the host names of the servers hosting the integration nodes.
8. Restart the integration servers for the changes to take effect.

   When they are restarted, `IntegrationServer1` and `IntegrationServer2` will be both catalog servers and container servers. `IntegrationServer3` will be a container server.

What to do next

If you have only one catalog server and you stop it, the cache becomes unavailable and your data is lost. However, if you have multiple catalog servers, one of them can be stopped and restarted without losing data; your data is lost only if all catalog servers are stopped.

Optimizing the embedded global cache for use with different types of cache data

You can configure your embedded global cache to use different locking strategies for different cache maps, and you can configure the use of replica shards of a cache map for read access.

Before you begin

For more information about the default global cache topology, see "Data caching overview" on page 272.

About this task

The embedded global cache in IBM App Connect Enterprise supports the configuration of the following properties:

- Locking strategies: Using the appropriate locking strategy can reduce the associated computational work and help improve achievable throughput. The following table details the strategies that are supported and the situations in which the strategies might be used:

Locking strategy	Use case
Pessimistic (default)	Cache data is updated frequently from multiple sources.

Locking strategy	Use case
Optimistic	Cache data is updated from multiple sources but the same cache record is unlikely to be updated by two sources simultaneously.
None	Cache data is static or updated infrequently from a single source.

For more information about the locking strategies, see the WebSphere eXtreme Scale documentation, http://www.ibm.com/support/knowledgecenter/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/cxslockstrategy.html.

Note: The optimistic locking strategy that is used in IBM App Connect Enterprise uses the "Optimistic no versioning locking" configuration that is described in the WebSphere eXtreme Scale documentation.

- Read access to replica shards of a cache map. Reading from replica shards of a cache map can help to distribute workload amongst container servers. By default, data is read from only the primary shard of a cache map but you can configure the embedded global cache to support read access to replica shards. You might configure this property if your applications can tolerate the use of cache data that might be out-of-date. For more information about read access to replica shards of a cache map, see the WebSphere eXtreme Scale documentation, http://www.ibm.com/support/knowledgecenter/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/cxsreadreplx.html.

Procedure

If you want to configure locking strategies or read access to replica shards of a cache map, you must complete the following steps:

1. Create a copy of the following files from the *InstallDir*`\server\sample\globalcache` directory where *InstallDir* is the IBM App Connect Enterprise installation directory:
   - `objectgrid.xml`
   - `deployment.xml`
2. Open the `objectgrid.xml` file and add or edit the `backingMap name` entries in the `<objectGrid name="WMB">` section with patterns that match the names of your cache maps.
3. Open the `deployment.xml` file and add or edit the `map ref` entries in the `<objectgridDeployment objectgridName="WMB">` section with patterns that match the names of your cache maps.

   Note:

   - You must not remove the entry that matches the default cache map for the IBM App Connect Enterprise embedded global cache (`SYSTEM.BROKER.*`).
   - The `backingMap name` entries in the `objectgrid.xml` file must be the same as the `map ref` entries in the `deployment.xml` file.
4. Optional: If you want to configure locking strategies for cache maps in the embedded global cache, see “Configuring locking strategies for the embedded global cache” on page 306.
5. Optional: If you want to configure the embedded global cache to read data from a replica shard of a cache map, see “Configuring the embedded global cache to read data from replica shards of a cache map” on page 308.

Configuring locking strategies for the embedded global cache
You can configure your embedded global cache to use locking strategies to suit the types of data that you store in your cache.

Before you begin

Complete the following steps:

- Create the `objectgrid.xml` and `deployment.xml` files and configure the cache map names; for more information, see "Optimizing the embedded global cache for use with different types of cache data" on page 305.

About this task

You define the locking strategy for each cache map in your embedded global cache by configuring the `objectgrid.xml` file. You then configure the embedded global cache by updating the `server.conf.yaml` configuration file for the integration server to point to the `objectgrid.xml` file. The embedded global cache can use only one `objectgrid.xml` file.

Procedure

1. Open the `objectgrid.xml` file, and configure the locking strategy for each `backingMap` entry by completing one of the following steps:

   - To configure the pessimistic locking strategy for a cache map, set the **`lockStrategy`** parameter to `PESSIMISTIC`.

     For example:

     ```
     <backingMap name="USER.PESSIMISTIC.*" template="true" timeToLive="0"
      ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="PESSIMISTIC" copyMode="COPY_TO_BYTES"/>
     ```

   - To configure the optimistic locking strategy for a cache map, set the **`lockStrategy`** parameter to `OPTIMISTIC_NO_VERSIONING`. You can also set the **`nearCacheInvalidationEnabled`** parameter to `true`.

     For example:

     ```
     <backingMap name="USER.OPTIMISTIC.*" template="true" timeToLive="0"
      ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="OPTIMISTIC_NO_VERSIONING"
      nearCacheInvalidationEnabled="true" copyMode="COPY_TO_BYTES"/>
     ```

   - To configure no locking for a cache map, set the **`lockStrategy`** parameter to `NONE`. You can also set the **`nearCacheInvalidationEnabled`** parameter to `true`.

     For example:

     ```
     <backingMap name="USER.NONE.*" template="true" timeToLive="0"
      ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="NONE" nearCacheInvalidationEnabled="true"
      copyMode="COPY_TO_BYTES"/>
     ```

   Note: Enabling near-cache invalidation removes stale data from the cache as quickly as possible. For more information, see the WebSphere eXtreme Scale documentation, https://www.ibm.com/support/knowledgecenter/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/txsnearcacheinv.html.

   The following example file shows three entries, and includes an entry (`SYSTEM.BROKER.*`) that matches the default cache map for the IBM App Connect Enterprise embedded global cache.

   Note: You can modify the locking strategy for the `SYSTEM.BROKER.*` entry but you must not remove the entry.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
     xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
     <objectGrid name="WMB">
      <backingMap name="USER.OPTIMISTIC.*" template="true" timeToLive="0"
    ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="OPTIMISTIC_NO_VERSIONING"
    nearCacheInvalidationEnabled="true" copyMode="COPY_TO_BYTES"/>
      <backingMap name="USER.PESSIMISTIC.*" template="true" timeToLive="0"
    ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="PESSIMISTIC" copyMode="COPY_TO_BYTES"/>
      <backingMap name="USER.NONE.*" template="true" timeToLive="0"
    ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="NONE" nearCacheInvalidationEnabled="true"
    copyMode="COPY_TO_BYTES"/>
   ```

```
   <backingMap name="SYSTEM.BROKER.*" template="true" timeToLive="0"
 ttlEvictorType="LAST_UPDATE_TIME" lockStrategy="PESSIMISTIC" copyMode="COPY_TO_BYTES"/>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

Note: If you add any additional `backingMap` entries, you must add the equivalent entries in the `deployment.xml`; see "Optimizing the embedded global cache for use with different types of cache data" on page 305. You cannot configure any other options in this file.

2. Save the `objectgrid.xml` file.
3. Configure the location of the `objectgrid.xml` file, by modifying the **objectGridCustomFile** property in the `server.conf.yaml` configuration file for the integration server.

   If you made changes to the `deployment.xml` file to keep the file consistent with the `objectgrid.xml`, you must also complete this step for your copy of the `deployment.xml` file. For information about the equivalent commands, see "Configuring the embedded global cache to read data from replica shards of a cache map" on page 308.
4. Restart the integration server for the changes to take effect.

Results

You have configured the IBM App Connect Enterprise embedded global cache to use one or more locking strategies.

Configuring the embedded global cache to read data from replica shards of a cache map

You can configure your embedded global cache to read data from replica shards of a cache map instead of the primary cache map.

Before you begin

Complete the following steps:

- Read the information about reading data from replica shards and the applications for which it is appropriate, in "Optimizing the embedded global cache for use with different types of cache data" on page 305.
- Create the `objectgrid.xml` and `deployment.xml` files and configure the cache map names, as described in "Optimizing the embedded global cache for use with different types of cache data" on page 305.

About this task

If more than one container server exists, the default cache policy ensures that all data is replicated at least once, so that each container server can host primary and replica shards of the cache data. By default, data is read only from the primary shard but you can configure the embedded global cache to support read access to replica shards.

You define that your embedded global cache can read data from replica shards of a cache map by configuring the `deployment.xml` file. You then configure the embedded global cache by updating the `server.conf.yaml` configuration file for the integration server to point to the `deployment.xml` file. The embedded global cache can use only one `deployment.xml` file.

Procedure

1. Open the `deployment.xml` file, and set the **replicaReadEnabled** parameter to `true` on the mapSet element.
   For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../
deploymentPolicy.xsd"
   xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
```

```
   <objectgridDeployment objectgridName="WMB">
      <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0" maxSyncReplicas="1"
 replicaReadEnabled="true">
         <map ref="USER.OPTIMISTIC.*"/>
         <map ref="USER.PESSIMISTIC.*"/>
         <map ref="USER.NONE.*"/>
         <map ref="SYSTEM.BROKER.*"/>
         <zoneMetadata>
            <shardMapping shard="P" zoneRuleRef="wmbRule"/>
            <shardMapping shard="S" zoneRuleRef="wmbRule"/>
            <zoneRule name="wmbRule" exclusivePlacement="false">
               <zone name="WMBZone"/>
            </zoneRule>
         </zoneMetadata>
      </mapSet>
   </objectgridDeployment>
</deploymentPolicy>
```

Note: If you add any additional `map` entries, you must add the equivalent entries in the `objectgrid.xml`; see "Optimizing the embedded global cache for use with different types of cache data" on page 305. You cannot configure any other options in this file.

2. Save the `deployment.xml` file.
3. Configure the location of the `deployment.xml` file in the integration server by modifying the **deploymentPolicyCustomFile** property in the `server.conf.yaml` configuration file.

   If you made changes to the `objectgrid.xml` file to keep the file consistent with the `deployment.xml`, you must also complete this step for your copy of the `objectgrid.xml` file. For information about the equivalent commands, see "Configuring locking strategies for the embedded global cache" on page 306.
4. Restart the integration server for the changes to take effect.

Results

You have configured read access to replica shards of one or more cache maps in the IBM App Connect Enterprise embedded global cache.

Connecting to external WebSphere eXtreme Scale grids

About this task

For information about WebSphere eXtreme Scale grids, see the concept topic "WebSphere eXtreme Scale grids" on page 278.

IBM App Connect Enterprise supports the IBM eXtremeIO (XIO) transport mechanism for communicating with WebSphere eXtreme Scale external grids. You do not need to specify this transport mechanism as part of the integration server configuration.

For detailed information about how to connect to an external WebSphere eXtreme Scale grid, and how to enable SSL on those connections, see the following topics:

- "Connecting to a WebSphere eXtreme Scale grid" on page 309
- "Enabling SSL for external WebSphere eXtreme Scale grids" on page 310

Connecting to a WebSphere eXtreme Scale grid

You can store data in a WebSphere eXtreme Scale grid for use by other message flows.

Before you begin

- Read the concept information about external grids in "WebSphere eXtreme Scale grids" on page 278.

- Ask your WebSphere eXtreme Scale administrator for the following information about the external grid to which you are connecting:
  - The name of the grid
  - The host name and port of each catalog server for the grid
  - Optional: The object grid file that can be used to override WebSphere eXtreme Scale properties
- To enable SSL for connections to external WebSphere eXtreme Scale grids, follow the instructions in "Enabling SSL for external WebSphere eXtreme Scale grids" on page 310.

Procedure

To connect to a WebSphere eXtreme Scale grid by using a policy, complete the following steps.

1. Optional: If you are connecting to a secure grid, use the **mqsisetdbparms** command to create a security identity, as shown in the following example.

   ```
   mqsisetdbparms INODE -n wxs::id1 -u userId -p password
   ```

2. Create a WXS Server policy by using the Policy editor in the IBM App Connect Enterprise Toolkit (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
3. Use the values that are provided by your WebSphere eXtreme Scale administrator to configure the connection to the external grid.

   When you specify the security identity, omit the prefix (wxs::) that you used when creating the security identity with the **mqsisetdbparms** command.

   If the object grid file is on a shared drive on Windows, you must use the `\\hostname\directory` path syntax to the shared drive, instead of a mapped drive letter. The IBM App Connect Enterprise user ID that is used to access the `\\hostname\directory` path must have read access to the file system and must use the same password that is required to access the shared drive.

What to do next

After you create the WXS Server policy, you can interact with the external grid by using a Mapping node or a JavaCompute node (see Accessing the global cache by using a Mapping node or "Accessing the global cache by using a JavaCompute node" on page 1781).

When you get a global map from an external grid, the getGlobalMap method makes a connection to the grid if one does not exist. To modify a flow that previously used the embedded cache to connect to an external grid instead, you must update your Mapping node or JavaCompute node by specifying the name of the map on the external grid and the name of the policy that is used to connect to the grid. You can work with multiple external grids, and the embedded grid, at the same time.

Statistics are recorded for the interactions between message flows and the external grid. The global cache activity log also provides a high-level overview of how IBM App Connect Enterprise interacts with the external grid. For more information, see "Monitoring the global cache" on page 313.

Enabling SSL for external WebSphere eXtreme Scale grids

Enable SSL for an external WebSphere eXtreme Scale grid by setting up a public key infrastructure, then enabling SSL on the integration server.

Before you begin

Read the concept information in "WebSphere eXtreme Scale grids" on page 278 and "Public key cryptography" on page 2492.

About this task

You can enable SSL for client connections to external WebSphere eXtreme Scale grids. You cannot enable SSL for servers in the embedded global cache.

To enable SSL communication, configure the keystore, truststore, passwords, and certificates. To enable server authentication, import the public certificate from the WebSphere eXtreme Scale server into the integration server truststore. If the server requires client authentication, you must also create a private key in the integration server keystore that the WebSphere eXtreme Scale server trusts.

You then modify the `server.conf.yaml` configuration file for the integration server and set properties to enable SSL and specify the required protocol. You can also nominate a particular key to use if you have more than one. SSL connections can be made only from integration servers that are not hosting catalog or container servers.

The following steps describe how to enable SSL for an external WebSphere eXtreme Scale grid.

Procedure

1. Set up a public key infrastructure by following the instructions in "Setting up a public key infrastructure" on page 2635.

   You can set up the public key infrastructure at integration node or integration server level.

   Connections to external WebSphere eXtreme Scale grids cannot implicitly use public certificates that are located in the JVM cacerts file.

Modify the properties in the `server.conf.yaml` configuration file for the integration server:

2. Use a YAML editor to open the `server.conf.yaml` file.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your integration server configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
3. Ensure that you are enabling SSL on an integration server that does not host a catalog or container server, by setting the **`enableCatalogService`** and **`enableContainerService`** properties in the `server.conf.yaml` configuration file to `false`.
4. To enable SSL, set the following properties in the `server.conf.yaml` file:
   - To enable SSL, set the **`clientsDefaultToSSL`** property `true`.
   - To specify an SSL protocol, set **`sslProtocol`** to a value that is recognized by the IBM JSSE2 security provider.
   - If the external grid requires client authentication and you have more than one trusted private key in the integration node keystore, set **`sslAlias`** to the appropriate key.
5. Restart the integration server for the changes to take effect.
6. Connect to the WebSphere eXtreme Scale grid by following the instructions in "Connecting to a WebSphere eXtreme Scale grid" on page 309.

Results

Keystore, truststore, and protocol settings are verified the first time that a connection is made from the integration server (either to the embedded grid, or for the first remote connection). Errors in the configuration are reported as a warning, and SSL connections are then prohibited. For example, a warning is issued if a keystore file is not found, the file is corrupted, or the keystore password is incorrect.

If you enable SSL and try to connect from an integration server that hosts WebSphere eXtreme Scale server components, the connection fails with a detailed exception message, BIP7144, which explains why the connection failed. If an SSL handshake exception occurs, the message flow fails and the exception message BIP7147 is issued.

Migrating global cache configuration settings

You can migrate your global cache settings by completing some additional configuration as part of your migration from IBM Integration Bus V10 to IBM App Connect Enterprise V11.

When you migrate from IBM Integration Bus V10 to IBM App Connect Enterprise V11, you can migrate your global cache configuration settings by completing the following steps, in addition to those described in the Chapter 2, “Migrating,” on page 19 section:

- “Migrating settings from your IBM Integration Bus cache policy XML files” on page 312
- “Migrating settings from your IBM Integration Bus objectgrid.xml and deployment.xml files” on page 313

Migrating settings from your IBM Integration Bus cache policy XML files

In IBM Integration Bus V10, cache policy XML files were used to configure the global cache. In IBM App Connect Enterprise, the properties that were set in those files are now set in the `server.conf.yaml` configuration files instead.

The following example shows a policy XML file that is similar to the sample `policy_two_brokers_ha.xml` file that was provided with IBM Integration Bus Version 10:

```
<cachePolicy xmlns="http://www.ibm.com/xmlns/prod/websphere/messagebroker/globalcache/
policy-1.0">
    <broker name="integrationNode1" listenerHost="localhost">
        <catalogs>1</catalogs>
        <portRange>
            <startPort>2800</startPort>
            <endPort>2803</endPort>
        </portRange>
    </broker>

    <broker name="integrationNode2" listenerHost="localhost">
        <catalogs>1</catalogs>
        <portRange>
            <startPort>2804</startPort>
            <endPort>2815</endPort>
        </portRange>
    </broker>
</cachePolicy>
```

In IBM Integration Bus Version 10, if you applied the example cache policy XML file to two integration nodes called `integrationNode1` and `integrationNode2`, which were running on the same host (in this example, `localhost`) with several integration servers, your integration nodes would have the following configuration as a result:

- Integration Node 1:
  - Integration Server 1: one catalog server and one container server (taking ports 2800-2803)
  - Further integration servers would have no catalog servers or container servers
- Integration Node 2:
  - Integration Server 2: one catalog server and one container server (taking ports 2804 - 2807)
  - Integration Server 3: container server (taking ports 2808 - 2811)
  - Integration Server 4: container server (taking ports 2812 - 2815)
  - Further integration servers would have no catalog or container servers

In IBM App Connect Enterprise, equivalent sample `server.conf.yaml` configuration files are provided under the sample directory `basic_2_catalogs_4_containers`, in which `IntegrationServer1` would be under `IntegrationNode1`, and `IntegrationServer2`, `IntegrationServer3`, and `IntegrationServer4` would be under `IntegrationNode2`.

This configuration is static and persists across integration server restarts, which means that an integration server that is configured as a catalog server will always be a catalog server, and you can tune your integration servers accordingly. For example, this might include giving container servers a larger JVM heap size, because this is where your cache is stored.

Migrating settings from your IBM Integration Bus objectgrid.xml and deployment.xml files

In IBM Integration Bus V10, the `objectgrid.xml` and `deployment.xml` files had to be copied into set locations beneath your integration node and integration server workpaths, before they would take effect. In IBM App Connect Enterprise, you set the **objectGridCustomFile** and **deploymentPolicyCustomFile** properties in the `server.conf.yaml` configuration file to point to these files. You can choose to put the files in the same workpath locations as in IBM Integration Bus V10, but, if you do so, you must specify the custom file properties in the `server.conf.yaml` file to point to this location.

Monitoring the global cache

You can use the **mqsicacheadmin** command, resource statistics, and the activity log to monitor the global cache. You can use resource statistics and the activity log to monitor external grids.

About this task

The following tools provide information about the global cache and how it is performing.

mqsicacheadmin command
The **mqsicacheadmin** command provides information about the global cache that is embedded in an integration server. For example, you can find out the size of a map and list the hosts that are participating in the cache. You can also use this command to clear data from a map. See "Running the mqsicacheadmin command" on page 313.

Activity log
Activity logs provide a high-level overview of how IBM App Connect Enterprise interacts with external resources, therefore helping you to understand what your message flows are doing. See "Viewing Activity Logs for message flows in the web user interface" on page 2856.

Resource statistics
Resource statistics are collected by an integration server to record performance and operating details of the resources that it uses. See "Managing resource statistics collection" on page 2741.

Running the `mqsicacheadmin` command

Procedure

To use the command to find information about the global cache that is embedded in an integration server, complete the following steps.

1. Before you run the **mqsicacheadmin** command, ensure that the integration server is running and that the global cache is available.

   The cache is disabled by default. To use the cache, set the properties in the Global Cache section of the `server.conf.yaml` configuration file for the integration server. For more information, see "Configuring the embedded global cache" on page 285.

2. Run the **mqsicacheadmin** command, specifying the appropriate parameters, as described in command. Use the **-c** parameter to run a WebSphere eXtreme Scale command.

   For example, to list all container servers and their shards in the embedded cache, run the WebSphere eXtreme Scale command, **showPlacement**:

   ```
   mqsicacheadmin myIntegrationNode -c showPlacement
   ```

   To clear all data from the map called *myMap* in the integration node *myIntegrationNode*, run the following command:

   ```
   mqsicacheadmin myIntegrationNode -c clearGrid -m myMap
   ```

   The data is cleared immediately; you do not need to restart the integration node for this action to take effect.

   You do not have to specify an integration node name with this command. For example, use the **-cep** parameter to connect to a catalog server and show the routing table for each WebSphere eXtreme Scale shard:

   ```
   mqsicacheadmin -cep server.company.com:2800 -c routetable
   ```

   The **-cep** parameter specifies the catalog service endpoints, and you can use this parameter to specify a comma-separated list of catalog servers; the first valid server to be contacted will be used.
3. Examine the output and usage information that is returned by the command.

Listing database connections

IBM App Connect Enterprise does not provide an interface that you can use to list the connections that it has to a database. You must use the facilities of the database suppliers to list connections.

For further information about how to list database connections, refer to the documentation that is provided by your database vendor.

Quiescing a database

The integration node and database exhibit specific behaviors when you quiesce the database.

If you access databases from one or more message flows, your database administrator might occasionally want to issue the quiesce instruction on a database. This action is a function of the database, not of the integration node.

The following three assumptions are made for the database that you are quiescing:

- The database can be quiesced (not all databases support this function).
- New connections to the database are blocked by the database when it is quiescing.
- Message flows that access the database eventually become idle.

The following list shows the behavior that is expected while a database is quiescing:

- Run the command that quiesces the database. When this command starts, the connections that are in use remain in use, but no new connections to the database are allowed.
- Messages that are being processed by message flows, which use existing connections to the database, continue to use their connections until the connections become idle. Therefore if messages continue to be received by the message flow, it might be a long time before the quiesce occurs. To ensure that messages are no longer processed, stop the message flow. Stopping the message flow stops messages being processed, and releases the database connections that the flow was using. This action ensures that the database connections that the flow holds become idle.
- Database connections for the message flow become idle. This situation causes the integration node to release the connections to the databases that the message flow is using. When all connections to the

database from the integration node, and from any other applications that are using the database, are released, the database can complete its quiesce function.

For more information, see "Database connections" on page 1131.

Using a JDBC connection pool to manage database resources used by an integration server

Use integration node JDBC provider resources to configure the use of thread pools independently of message flow and input node thread pools.

About this task

IBM App Connect Enterprise manages JDBC connections in the following ways:

- Non-pooled connections:
  - IBM App Connect Enterprise creates a JDBC connection on demand for each message flow instance that requires one.
  - Each JDBC connection is associated with the message flow instance for which it was created. This association is maintained until the connection is closed.
  - Each JDBC connection that is idle for 60 seconds is closed, and is no longer associated with a message flow instance.
  - After a JDBC connection that was associated with a message flow instance is closed, if the same message flow instance requires a JDBC connection, IBM App Connect Enterprise creates a new JDBC connection on demand.
- Pooled connections:
  - When a message flow instance requires a JDBC connection, IBM App Connect Enterprise assigns an unused connection from the pool.
  - If all pooled JDBC connections are being used, and the maximum pool size has not been reached, IBM App Connect Enterprise creates a new pooled JDBC connection. The maximum pool size is specified in the `Maximum size of connection pool` property of the JDBC Providers policy (JDBCProviders).
  - Each pooled JDBC connection remains associated with a message flow instance only for the processing of one input message.
  - When a message flow instance completes the processing of an input message, the association with a JDBC connection is removed, and the JDBC connection is returned to the pool.
  - Each pooled JDBC connection that is idle for 15 minutes is closed, and is removed from the pool.
  - Pooled JDBC connections are not applicable to the DatabaseRetrieve and DatabaseRoute nodes.

Using a JDBC connection pool enables you to scale database access independently of the number of message flow threads.

The purpose of the JDBC connection pooling function in IBM App Connect Enterprise is to:

- Minimize the time and resource needed to create new connections, by making pooled connections available for reuse by other message flow threads when transactions complete.
- Enable greater control over the number of concurrent database connections that can be in use at any time for a specific JDBC data source.

Although these aims overlap with those of J2EE Java connection pooling, the priority is not to maximize the performance of individual JDBC calls to the database. In this respect, non-pooled JDBC connections might be more efficient in some scenarios.

You can create a JDBC connection pool by setting the `Maximum size of connection pool` property of the JDBC Providers policy to a non-zero integer value. This property acts at the integration server level to specify the maximum number of JDBC connection threads that can be used. A value of zero defaults to

the standard WebSphere Message Broker Version 8.0 behavior, where one JDBC connection is created for each message flow thread.

All message flows within an integration server that use the same JDBC Providers policy also share a connection pool. You can monitor the behavior of a JDBC connection pool by using integration node resource statistics

The `Maximum size of connection pool` property is applicable to JDBC connections obtained using the `getJDBCType4Connection()` API of the JavaCompute node, and to database operations in graphical data maps that are called by the Mapping node.

Note: The `Maximum size of connection pool` property does not apply to the JDBC connections used by the DatabaseRetrieve or DatabaseRoute nodes.

Managing deployed resources

You can manage the resources that you have deployed to integration servers by using commands, the IBM App Connect Enterprise Toolkit, or the web user interface. You can also use the IBM Integration API or commands to complete some of these actions.

About this task

- “Setting the start mode of message flows and applications at run time” on page 316
- “Starting or stopping deployed resources” on page 317
- “Viewing version information and custom keyword values in your deployed solutions” on page 319
- “Deleting deployed resources” on page 319
- “Managing a deployed REST API” on page 323

Setting the start mode of message flows and applications at run time

You can configure the run state of applications, integration services, and message flows when you deploy or when you restart an integration server.

Before you begin

This topic assumes that a BAR file has been created and contains an application, integration service or message flow. For more information, see the following topics:

- “Creating a BAR file” on page 2469
- “Adding resources to a BAR file” on page 2470

About this task

You can set the default behavior of message flows and applications when you deploy or restart an integration server. For example, you might have a message flow that creates resources that are required by other message flows. Therefore, you might want to start one message flow before all others when the flows are deployed, or when the integration server or containing application is started. You can set one message flow to start automatically, then set other message flows to require a manual restart.

You can configure the run state by using the `startMode` property on the **`mqsiapplybaroverride`** command. Use the **`mqsireadbar`** command to view the current values of properties in the BAR file. You can set the `startMode` property to one of the following values:

Maintained
: This value is the default, and indicates that the flow or application is started when deployment is complete, and remains running until a stop command is issued. After a stop command has been issued, the flow or application remains stopped until a start command is issued. The state of the flow or application remains unchanged after redeployment, or after the integration server or containing

application has been restarted. When an application holds one or more flows, the state of the flows within the application is preserved if the application is stopped and then started again.

Manual
This value indicates that the flow or application must always be started manually after deployment. or after the integration server or containing application has been restarted. The flow or application is in stopped state after deployment or redeployment, and after the integration server or containing application is restarted, even if the flow or application was running before the deployment or restart.

Automatic
This value indicates that the flow or application is always started automatically after deployment, redeployment, or after the integration server or containing application is restarted.

You can set this property for message flows and applications. The state of an application overrides the state of any message flows that it contains. For example, if an application is stopped, no flows in that application can run, even if they have been set to start automatically.

To find out how to use the **mqsiapplybaroverride** command to set the `startMode` property, see command.

What to do next

Deploy the BAR file by following the instructions in "Deploying integration solutions to a production environment" on page 2480.

Starting or stopping deployed resources

Use the IBM App Connect Enterprise Toolkit or the web user interface to start or stop applications, integration services, and message flows. Use the **mqsistart** or **mqsistop** commands to start or stop applications, libraries or message flows.

About this task

You cannot start or stop libraries or subflows. If a message flow includes a subflow that is defined in a `.subflow` file, you can start your message flow only if the subflow is deployed to the same integration server.

If you stop the integration server in which an application, integration service, or message flow is running then, by default, the mode of the resource (started or stopped) is retained. However, you can configure the resources so that, when you deploy or restart an integration server, the resources start in a specific mode; see "Setting the start mode of message flows and applications at run time" on page 316.

You can use the following methods to stop and start applications, libraries, integration services, and message flows:

- "Starting and stopping an application, integration service, or message flow by using the IBM App Connect Enterprise Toolkit" on page 317
- "Starting and stopping an application, integration service, or message flow by using the web user interface" on page 318
- "Starting and stopping an application or a message flow by using the mqsistart or mqsistop commands" on page 318
- "Starting and stopping an application or a message flow by using the mqsistartmsgflow or mqsistopmsgflow commands" on page 318

Starting and stopping an application, integration service, or message flow by using the IBM App Connect Enterprise Toolkit

Procedure

1. In the Integration Explorer view, expand the integration server.

2. Right-click the application, integration service, or message flow:
   - To start the resource, click **Start**.
   - To stop the resource, click **Stop**.

Results

A message is sent to the integration server to start or stop the resource.

Starting and stopping an application, integration service, or message flow by using the web user interface

Procedure

1. Start the IBM App Connect Enterprise web user interface, as described in "Accessing the web user interface" on page 89.

   Your integration servers, message flows, and other resources are displayed.
2. Navigate to the resource that you want to work with, select it, and then click **Start** or **Stop**.

Results

A message is sent to start or stop the applications, integration services, and message flows that you specified.

Starting and stopping an application or a message flow by using the `mqsistart` or `mqsistop` commands

Procedure

In the Command Console, run the **`mqsistart`** or **`mqsistop`** commands with the appropriate parameters for the deployed resource you want to start or stop. For more information, see command and command.

Results

A message is sent to the integration server to start or stop the resource.

Starting and stopping an application or a message flow by using the `mqsistartmsgflow` or `mqsistopmsgflow` commands

Procedure

In the Command Console, run the **`mqsistartmsgflow`** or **`mqsistopmsgflow`** commands with the appropriate parameters for the deployed resource you want to start or stop. For more information, see command and command.

Results

A message is sent to the integration server to start or stop the resource.

Viewing version information and custom keyword values in your deployed solutions

You can view version and custom keyword information in your deployed resources.

About this task

If version or custom keyword information was added to your resources during development, you can view the values in resources that are deployed to an integration server by using the following methods:

- "Viewing version and custom keyword information by using the IBM App Connect Enterprise Toolkit" on page 319

Viewing version and custom keyword information by using the IBM App Connect Enterprise Toolkit

Procedure

To view the version and custom keyword information that is associated with the resources that are deployed to an integration server by using the IBM App Connect Enterprise Toolkit, complete the following steps:

1. In the **Integration servers** pane of the IBM App Connect Enterprise Toolkit, locate the integration server where your resources are deployed.
2. Click a deployed resource to see the properties that are associated with the resource.

Results

If the resource has a version or custom keyword value, the values are displayed in the **Keywords** section of the **Properties** tab.

What to do next

For information about adding version or custom keyword information to your development resources, or viewing version or custom keyword information in your packaged resources, see the following topics:

- "Adding version information and custom keyword values to your development resources" on page 2466
- "Viewing version information and custom keyword values in your packaged solutions" on page 2468

Deleting deployed resources

Use the IBM App Connect Enterprise Toolkit or the web user interface to remove deployed integration solution resources such as applications and libraries from an integration server.

About this task

You can delete applications, libraries, integration services, and message flows from an integration server by using any of the following methods:

- "Deleting a deployed resource by using the web user interface" on page 320
- "Removing a deployed object by using the IBM App Connect Enterprise Toolkit" on page 320
- "Removing a deployed object by using the mqsideploy command" on page 320
- "Removing a deployed object by using the IBM Integration API" on page 321

Note: You cannot delete message flows or other resources that are contained in an integration service, application, or library.

You cannot delete shared libraries if they are being referenced by any of the other resources that are deployed on the integration server.

You cannot delete policy projects that contain non-dynamic policies. The changing or deletion of policies within a policy project is only supported for certain types of policy. To delete a policy project, which contains non-dynamic policies, it is necessary to delete all deployed resources from the integration server and then redeploy all resources.

For more information about policy properties, see Policy properties.

Deleting a deployed resource by using the web user interface

Procedure

1. Start the IBM App Connect Enterprise web user interface, as described in "Accessing the web user interface" on page 89.

   Your servers (integration servers), message flows, and other resources are displayed.
2. Select the resource that you want to delete and click **Delete**.

Results

A message is sent to the integration server to delete the selected resource.

Removing a deployed object by using the IBM App Connect Enterprise Toolkit

About this task

To remove an object from an integration server by using the IBM App Connect Enterprise Toolkit, complete the following steps.

Procedure

1. In the Integration Explorer view, right-click the object that you want to remove.
2. Click **Delete**, then **OK** to confirm.

Results

The request is sent to the integration server, and a synchronous response is sent back.

Removing a deployed object by using the `mqsideploy` command

About this task

To remove an object from an integration server by using the **mqsideploy** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.
2. Enter the appropriate command for your operating system and configuration, by using the following examples as a guide.

```
mqsideploy -i ipAddress -p port -e integrationServerName
-d file1.cmf:file2.bar:file3.dictionary:file4.xml
```

```
mqsideploy --admin-host ipAddress --admin-port port -e integrationServerName
-d file1.cmf:file2.bar:file3.dictionary:file4.xml
```

   The **-i** (IP address) and **-p** (port) parameters represent the connection details for an independent integration server or an integration node. If you want to run this command against an integration

node on the same computer, you can specify the integration node name instead, as in the following example:

```
mqsideploy integrationNodeName -e integrationServerName
-d file1.cmf:file2.bar:file3.dictionary:file4.xml
```

If you have a `.broker` file that contains the connection details for an independent integration server or integration node, you can specify this file by using the **-n** parameter, as in the following example:

```
mqsideploy -n integrationNodeFileName -e integrationServerName
-d file1.cmf:file2.bar:file3.dictionary:file4.xml
```

where *integrationNodeFileName* is the path and file name of the `.broker` file.

The **-d** parameter is a colon-separated list of files that you want to remove from the named integration server. When you run the command, the deployed objects (`file1.cmf, file2.bar, file3.dictionary, file4.xml`) are removed from the specified integration server.

Optionally, specify the **-m** parameter to remove all currently deployed message flows and message sets from the integration server as part of the deployment. If you do not set **-m**, the contents of the BAR file are deployed in addition to what is already deployed to the integration server. Any deployed objects with the same name as an item inside the BAR file are replaced by the version inside the BAR file. When you remove a message flow or message set, the **-m** parameter is ignored.

For more information about alternative long names for parameters, and for more information about the command, see command.

Results

The command reports when responses are received from the integration node. If the command completes successfully, a zero return code (0) is displayed.

Removing a deployed object by using the IBM Integration API

About this task

To remove deployed objects from an integration server, get a handle to the relevant `ExecutionGroupProxy` object, then run the `deleteDeployedObjectsByName` method. Use the following code as an example.

```
import com.ibm.broker.config.proxy.*;

public class DeleteDeployedObjects {
  public static void main(String[] args) {
    BrokerConnectionParameters bcp =
            new IntegrationNodeConnectionParameters
                      ("localhost", 4414);
    try {
      BrokerProxy b =
            BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy e =
            b.getExecutionGroupByName("default");
      e.deleteDeployedObjectsByName(
            new String[] { "file1.msgflow",
                           "file2.msgflow",
                           "file3.xsd",
                           "file4.msgflow" }, 0);
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

What to do next

If you remove one or more message flows, you can now remove the resource files that are associated with those message flows; for example, JAR files.

Deleting deployed policies

You can delete only policy projects that contain dynamic policies. To delete a policy project that contains non-dynamic policies, you must delete all deployed resources from the integration server.

You can delete the following types of policy after they are deployed. When you redeploy the policy project, all message flows that are using the policy are stopped and restarted:

- ActivityLog
- Aggregation
- CDServer
- CICSConnection
- Collector
- CORBA
- EmailServer
- FtpServer
- IMSConnect
- JDEdwardsConnection
- Kafka
- MQEndpoint
- MQTTPublish
- MQTTSubscribe
- ODMServer
- PeopleSoftConnection
- Resequence
- SAPConnection
- SecurityProfiles
- SiebelConnection
- SMTP
- TCPIP Client policy (TCPIPClient)
- TCPIP Server policy (TCPIPServer)
- Timer
- UserDefined
- WorkloadManagement

To delete the following policies after they are deployed, you must delete all deployed resources from the integration server:

- DotNet Application Domain policy (DotNetAppDomain)
- Java Class Loader policy (JavaClassLoader)
- JDBC Providers policy (JDBCProviders)
- JMS Providers policy (JMSProviders)
- Monitoring profiles
- WXSServer policy

For more information about policy properties, see Policy properties.

Managing a deployed REST API

After you deploy a REST API, you can manage it from the command line or the web user interface.

Before you begin

You must create a REST API in the IBM App Connect Enterprise Toolkit, see "Creating a REST API" on page 2019 and "Packaging and deploying a REST API" on page 2045.

You can view the REST API after it is deployed by using the web user interface, see web user interface.

About this task

After you deploy the REST API, you can manage it from the command line or the web user interface. When you are using the command line to manage a deployed REST API and the command you are using requires the name of an application, you must specify the name of the REST API. For example:

```
mqsilist integrationNodeName -e integrationServerName -k restApiName -d2
```

You can view the REST API after it is deployed by using the web user interface, see web user interface.

Procedure

To view information about the deployed REST API in the web user interface, complete the following steps:

1. Open the web user interface in a web browser. Locate the deployed REST API in the navigator.

   The REST API is listed under the integration server, in the REST APIs category.
2. Expand the navigation of the deployed REST API.

   The currently deployed message flow, subflows, and any other deployed resources are displayed.
3. Click the drop-down arrow for the deployed REST API.

   A list of actions that you can perform on a deployed REST API displayed. Depending on your access rights, you can start, stop, or delete the REST API, or start or stop statistics for the REST API.
4. Click the deployed REST API.

   The **Overview** tab for the deployed REST API is displayed.

   In the **Quick View** section of the **Overview** tab, the base URL for REST API invocations is displayed. The REST API base URL value contains the scheme (`http://` or `https://`), hostname, port number, and base path of the REST API that is running. By using an HTTP client, you can use this URL, along with other details of the operation that is being called, to call an operation in the deployed REST API.

   In the **Quick View** section of the **Overview** tab, the URL for the REST API definitions is displayed. You can use the REST API definitions URL to access the Swagger document or the OpenAPI 3.0 document for the deployed REST API. You can use this URL with Swagger tools, such as Swagger UI or Swagger Codegen or OpenAPI 3.0 tools. The Swagger document or OpenAPI 3.0 document available at this URL is automatically updated to contain the correct scheme (`http://` or `https://`), hostname , port number, and base path of the REST API that is running. Therefore, you can use the Swagger document or OpenAPI 3.0 document without modification.
5. Click the **API** tab.

   The details of resources and operations within the deployed REST API are displayed. Operations are displayed with their HTTP method, name, description, and information about whether that operation is implemented as a subflow.
6. Click an operation.

   The parameters for that operation are displayed, along with their name, type, description, and whether a parameter is required.

Results

The information about a deployed REST API is displayed.

Overriding properties at run time with policies

Use policies to control the behavior of message flows, and the nodes in message flows, at run time.

Before you begin

For more information about policies, read "Policies overview" on page 324.

About this task

By taking advantage of policies, you can define a common approach to controlling particular node properties, such as connection credentials, and certain aspects of message flow behavior, including flow rate. You can create and update policies at any time in the solution lifecycle.

Procedure

1. Create a policy by using the Policy editor and, if appropriate, attach it to a message flow node (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).

   Policies are XML files; therefore, you can create a policy by using an XML editor if you do not want to use the Toolkit. Policy documents are defined by an XML Schema Definition (`Policy.xsd`), which is found in the default installation directory. For example, on Windows, you can find the policy definition file at `C:\Program Files\IBM\ACE\version\common\schemas\Policy`.
2. Deploy the policy in a policy project.

   You can either deploy the policy project in an independent BAR file or in the same BAR file as the associated message flow (see "Adding resources to a BAR file" on page 2470).

   Deployed policies are shown in the `run` sub-directory of the integration server work directory, within a sub-directory with the name of the policy project. You can see the properties of deployed policies in the Integration Explorer of the IBM App Connect Enterprise Toolkit or on the Policies tab of the web user interface (see "Accessing the web user interface" on page 89). You can also see which properties are being used by the deployed flow at run time by using the REST API or web user interface.
3. Optional: You can override a deployed policy by putting an updated policy project in the `overrides` sub-directory of the integration server work directory (see "Overriding deployed policies at run time with the overrides directory" on page 328).

Policies overview

Policies can control particular node properties, such as connection credentials, and certain aspects of message flow behavior, including flow rate. Policies provide a shared and managed definition that you can reuse.

Policies contain defined operational properties, and enable the following user capabilities:

- Developers can use policies to control runtime behavior and reuse configuration information in multiple places, therefore eliminating duplication of effort. Where policies are used to override message flow properties at run time, information like connection details can be updated without having to update message flows.
- Administrators can use policies to define key configuration data for each environment and to override properties that were specified by message flow developers. This ability is useful if connection details are different for different environments (such as development, test, and production systems.) Administrators can also update policies with sensitive data that might not be available to developers.
- Operators can use policies to monitor and modify configuration data dynamically.

You can attach a policy to one or more entities that you want to control, such as message flows, message flow nodes, or integration servers. For example, if you add an AggregateControl node to your flow, the

`Aggregate name` property on the message flow node can be used to associate fan-in and fan-out flows. Alternatively, it can specify the name of an Aggregation policy. If you specify just the name of the policy in the `Aggregate name` property, the policy is retrieved from the default policy project. Alternatively, you can specify a policy in a different policy project by using the following format in the message flow node property:

```
{PolicyProjectName}:PolicyName
```

If the named policy cannot be found, the message flow cannot be started.

You can also reference a policy from a JavaCompute node by using the name of the policy in the Java code; for example,

```
MbPolicy myUserPolicy = getPolicy("UserDefined", "{MyPolicyProject}:MyUserPolicy");
```

Policies are XML documents, which are defined by an XML Schema Definition (`Policy.xsd`) that is found in the default installation directory. For example, on Windows, you can find the policy definition file at `C:\Program Files\IBM\ACE\`*`version`*`\common\schemas\Policy`.

Configurable services

Policies replace configurable services in App Connect Enterprise. You can migrate your existing configurable services to policies by using the **mqsiextractcomponents** command (see command).

The **mqsicreateconfigurableservice** command is deprecated in App Connect Enterprise, and no longer creates configurable services. However, if you use the **mqsicreateconfigurableservice** command in your IBM Integration Bus scripts, this command creates equivalent policies (where supported) in App Connect Enterprise (see command (deprecated)).

Policies that are created by the **mqsicreateconfigurableservice** command are created in the default policy project (`DefaultPolicies`) and are associated with an integration node. These policies are inherited by all integration servers on that integration node. You can override a policy that is inherited by an integration server from the integration node by deploying an updated version of the policy to that integration server. Alternatively, you can put an updated policy in the `overrides` subdirectory of the integration server work directory (see “Order of precedence for overrides” on page 326).

Policy projects

When you run the **mqsicreateworkdir** command, a configuration file called `server.conf.yaml` is created, which contains default settings for your integration server. This configuration file defines a default policy project, which is used to store unqualified policies. When you create a policy in the IBM App Connect Enterprise Toolkit, you must create it in a policy project. When you reference a policy from a message flow or message flow node, use the policy name if the policy is in the default policy project. If the policy is in a different policy project, prefix the name of the policy with the name of the policy project, with the format *{policyProject}:PolicyName*. You can use policy projects to organize your resources, and you can add as many policies to a policy project as you want. For example, your policy project might organize policies in the following ways:

- Your policy project might contain all policies for a particular integration server.
- The project might contain all policies of a particular type (for example, all JDBC Providers policies) for an integration server.
- It might contain all policies for a specific application.

For more information about creating policies, see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327.

Default policy project and policies

A default policy project called `DefaultPolicies` is defined in the `server.conf.yaml` file for the integration server. Policies that are created by the **mqsicreateconfigurableservice** command are

created in this default policy project. You can also specify certain default policies to control resources on an integration server by setting them in the `server.conf.yaml` file for that integration server (see “Configuring an integration server by modifying the server.conf.yaml file” on page 172). For example, the following excerpt from the `server.conf.yaml` file shows that you can specify a default Monitoring Profile policy to configure monitoring for message flows. For more information, see “Monitoring profiles” on page 2775.

```
Defaults:
  #defaultApplication: ''            # Name a default application under which independent
 resources will be placed
  #policyProject: 'DefaultPolicies'  # Name of the Policy project that will be used for
 unqualified Policy references. Default is 'DefaultPolicies'
  Policies:
    # Set default policy names, optionally qualified with a policy project as {policy
 project}:name
    #monitoringProfile: ''           # Default Monitoring profile
```

After you create the policy, set the name of the policy in the `server.conf.yaml` file. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project (*{policyProject}:PolicyName*).

Policy deployment

You deploy policies in policy projects in an independent BAR file, or in a BAR file with the associated message flow. As an administrator, you can also override deployed policies by using the `overrides` subdirectory of the integration server. For more information, see “Overriding deployed policies at run time with the overrides directory” on page 328. If a policy name is set specifically on a message flow or message flow node, you must deploy the policy before the message flow can be started. If a policy is not specified explicitly on a message flow or message flow node, you must deploy the policy before or with the message flow.

You can change and redeploy policies after they are deployed, by redeploying the policy project. When you redeploy a dynamic policy, all message flows that are using it are stopped and restarted. For non-dynamic policies, which cannot be deployed while applications and message flows are using them, you must restart all applications as part of the deploy operation by specifying the **--restart-all-applications** parameter on the **mqsideploy** or **ibmint deploy** command. For more information about the types of policy that can be redeployed, see Policy properties.

Deployed policies are shown in the `run` subdirectory of the integration server work directory, within a subdirectory that has the name of the policy project. You can see the properties of deployed policies in the Integration Explorer of the IBM App Connect Enterprise Toolkit or on the **Policies** tab of the web user interface (see “Accessing the web user interface” on page 89). You can also see which properties are being used by the deployed flow at run time by using the REST API or web user interface.

For more information about policy deployment, see "What to do next" in “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327.

Order of precedence for overrides

You can override a deployed policy by putting an updated policy project in the `overrides` subdirectory of the integration server work directory and restarting the integration server (see “Overriding deployed policies at run time with the overrides directory” on page 328).

The following precedence rules apply to operational property values:

- The properties in HTTP or HTTPS Connector policies completely override those properties that were set by the HTTP or HTTPS Connector in the ResourceManagers section of the `server.conf.yaml` file.
- A value in a policy in the overrides subdirectory of the work directory overrides a value in a deployed policy.

- A value in a deployed policy overrides a value in an integration node-level policy that is created by the **mqsicreateconfigurableservice** command .
- A value in an integration node-level policy takes precedence over a value that is overridden in a BAR file or set on the node.
- When no policy exists for an operational property value, the value that is overridden in a BAR file takes precedence over the value that is set on the node.
- When no policy or BAR file override exists for an operational property value, the value that is set on the node takes precedence.

Tip: If you designed your message flow to include a local environment override, any property value that is set by the local environment override overrides a value that is set by a policy, a BAR file override, or on the node.

Policies replace configurable services in IBM App Connect Enterprise 11.0. In previous versions, if you used a configurable service to override node properties at run time, a blank property in a configurable service overrode a node property with an empty value. However, when you use a policy, an empty property in a policy does not override the node property with an empty value. In this case, the value that is set by the node property is used.

Policy types

In addition to the policy types that are listed in Policy properties, monitoring profiles and policy sets and bindings are forms of policy that can be used to override properties at run time. For more information about these types of policy, see “Monitoring profiles” on page 2775 and “Policy sets” on page 2662.

Creating policies with the IBM App Connect Enterprise Toolkit

To control message flow behavior at run time, create one or more policies in one or more policy projects and deploy the policy projects.

Before you begin

For more information about policies, read “Policies overview” on page 324.

About this task

Policies are used to override property values so that you can control the behavior of message flows and message flow nodes at run time. You create one or more policies in a policy project, then you deploy one or more policy projects. Policies are XML files; therefore you can create a policy by using an XML editor. Policies are defined by an XML Schema Definition (`Policy.xsd`), which is found in the default installation directory. (For example, on Windows, you can find the policy definition file at `C:\Program Files\IBM\ACE\version\common\schemas\Policy`.) However, the following steps describe how to create a policy and a policy project in the IBM App Connect Enterprise Toolkit.

Procedure

1. In the Application Development view of the IBM App Connect Enterprise Toolkit, click **New**, then select **Policy**.
2. Select an existing policy project or click **New** to create a policy project.
3. Enter a name for your policy.

   The policy name can include alphanumeric characters and the underscore character. The name cannot include spaces and must start with a letter.
4. Click **Finish**.

   A `.policyxml` file is created and is opened in the Policy editor.
5. In the Policy editor, select the policy type from the list.
6. If your chosen policy type has more than one template, select the appropriate template from the list.

   The template provides default values for some fields.

7. Edit or set appropriate values for your policy.

   Mandatory properties are marked with an asterisk. For more information about property values for specific policies, see Policy properties.

8. Save your policy.

   Any errors are reported on the **Problems** tab.

Results

A policy that contains the overriding property values is generated. You can see your policy and policy project in the Application Development view.

The values in the policy override the equivalent properties that were configured in the **Properties** view for the node when the message flow was developed.

What to do next

1. Attach the policy to a message flow node by setting the policy name in the appropriate property of the message flow node.

   If you want to override a message flow node property with a policy at run time, you need to set the policy name on the message flow node. For example, to configure storage of events for aggregation nodes at run time, first create an Aggregation policy. Next, set the **Aggregate name** property on the AggregateControl or AggregateReply node to the name of the policy. If you do not use an Aggregation policy, the **Aggregate name** property on the Aggregation nodes is used to associated fan-in and fan-out flows.

   If the policy is deployed in the default policy project for the integration server, specify just the policy name in the message flow property.If the policy is deployed in a non-default policy project, prefix the name of the policy with the policy project name:

   ```
   {PolicyProjectName}:PolicyName
   ```

   You can also reference a policy from a JavaCompute node by using the name of the policy in the Java code; for example:

   ```
   Connection conn = getJDBCType4Connection("MyJDBCPolicy", JDBC_TransactionType.MB_TRANSACTION_AUTO);
   ```

2. Deploy the policy.

   You can deploy one or more policy projects, either in an independent BAR file, or in the same BAR file as the associated message flow.

   If a policy name is set specifically on a message flow or message flow node, you must deploy the policy before the message flow can be started. If a policy is not specified explicitly on a message flow or message flow node, you must deploy the policy before or with the message flow (see “Adding resources to a BAR file” on page 2470).

Deployed policies are shown in the `run` sub-directory of the integration server work directory, within a sub-directory with the name of the policy project. You can see the properties of deployed policies in the Integration Explorer of the IBM App Connect Enterprise Toolkit or on the Policies tab of the web user interface (see “Accessing the web user interface” on page 89). You can also see which properties are being used by the deployed flow at run time by using the REST API or web user interface.

Overriding deployed policies at run time with the overrides directory

You can override a deployed policy by putting an updated policy in the `overrides` subdirectory of the integration server work directory.

About this task

Policies that are deployed to an integration server are stored in the `run` directory for that integration server. Each integration server also has an `overrides` directory, which can contain policies that are

read by the integration server when it is starting up. If a policy in the overrides subdirectory has the same name as a policy that is deployed in a BAR file, the policy in the overrides subdirectory takes precedence.

Procedure

To override a deployed policy at run time, complete the following steps.

1. In the `overrides` subdirectory of the integration server, create a subdirectory with the name of the policy project that contains the policy that you want to override. For example, *myPolicyProject*.
2. Export the policy project that contains the policy that you want to update from the IBM App Connect Enterprise Toolkit to your file system. You need to include the file `policy.descriptor`.
3. Edit the appropriate property values in the `.monprofile.xml` file.
4. Save the updated `.policyxml` file and the `policy.descriptor` file to the subdirectory that you created in Step 1.
5. Restart the integration server.

Configuring a default policy project

You can configure a default policy project for an integration server by setting the policy project name in the integration server's `server.conf.yaml` configuration file.

Before you begin

- Read "Overriding properties at run time with policies" on page 324 for background information.
- Create a policy project for the required message flow nodes.
- Deploy your message flows to the integration server.

About this task

You can configure a default policy project to be used by an integration server, by setting the **`policyProjects`** property in the `Defaults` section of the integration server's `server.conf.yaml` configuration file, as shown in the following example. The policy project that you specify is then used when a policy is required but no policy name has been specified. The default value for the **`policyProject`** property is `DefaultPolicies`.

```
Defaults:
  #defaultApplication: ''            # Name a default application under which independent
 resources will be placed
  #policyProject: 'DefaultPolicies'  # Name of the Policy project that will be used for
 unqualified Policy references, default is 'DefaultPolicies'
  Policies:
    # Set default policy names, optionally qualified with a policy project as {policy
 project}:name
    #monitoringProfile: ''       # Default Monitoring profile
  Credentials:
    # Names a default credential name to be used when a more specific credential is not
 available for the credential type.
    #httpproxy: ''
    #jdbc: ''
    #kafka: ''
    #kerberos: ''
    #ldap: ''
    #odbc: ''
    #mq: ''
    #wsrr: ''
```

Procedure

To configure a default policy project for an integration server, complete the following steps:

1. Use a YAML editor to open the `server.conf.yaml` file.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your integration server configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Set the **`policyProject`** property in the `Defaults` section of the `server.conf.yaml` file, by removing the comment (`#`) from the start of the `policyProject` line and either specifying the name of an existing policy project or keeping the default value of `'DefaultPolicies'`, as shown in the following example:

   ```
   policyProject: 'DefaultPolicies'   # Name of the Policy project that will be used for
    unqualified Policy references
   ```
3. Restart the integration server. The properties that you set in the `server.conf.yaml` file take effect when the integration server is started. If you modify these properties again, you must also start the integration server again for the subsequent changes to take effect. For more information, see “Starting an integration server” on page 250.

Viewing administration activity in the admin log

App Connect Enterprise stores a log of administration activity for each integration node and integration server.

About this task

Admin logs are generated for administration activities that occur at the level of integration nodes, managed integration servers, and independent integration servers. At the integration node level, the admin log contains information about events such as starting and stopping integration servers. At the integration server level, the admin log contains information about the admin events that occur on that server. The logs contain information such as the date and time of an action, the description and result of the action (success or failure), and the username and authorized role of the user who initiated the action.

The admin log is enabled by default, and you can view the data either by selecting the **Admin Log** tab in the App Connect Enterprise web user interface, or by using the administration REST API. The web user interface shows the most recent data for the integration node or server since it started, including the BIP message number, message text, timestamp, username, and authorized role. It also shows a set of tags for each entry, which can be used to identify such things as the message flow, the application, and the action that was performed. Admin events that are held in the integration node or server in-memory buffer are loaded and displayed when the tab is first selected. To display events that have occurred since the tab was opened, click **Refresh**. The timestamp shows when the tab was last refreshed.

You can filter the entries that are displayed in the web user interface, as described in “Filtering admin log entries in the web user interface” on page 331.

In addition to displaying admin log entries in the web user interface, you can write admin log entries for an integration node or server to a file, as described in “Writing admin log entries to a file” on page 332. For independent integration servers, you can configure administration logging to write to the console. For information about configuring administration logging, see “Configuring administration logging” on page 333.

Admin log entries are created for administration actions that are made through all MQSI commands, regardless of whether the integration node or server is running. When a command is run against a running integration node or server, the admin log entries are made by the integration node or server. These admin log events are held in an in-memory list, which can be accessed through the administration REST API and viewed through the **Admin Log** tab of the web user interface. If the admin logging **`fileLog`** option has been enabled for the integration node or server, the entries are also written to the `AdminLog` file. When a command is run against an integration node or server that is not running, the command writes the

event to the AdminLog file, but it is not accessible to the administration REST API and cannot be viewed through the web user interface.

The username and role that are logged in an admin log entry depend on whether the request is made against a local or remote integration node or integration server (independent or managed), and on whether REST administration basic authentication or authorization is enabled:

- For command, toolkit, web user interface, and REST API requests issued to a *local* integration node or integration server (where *local* means on the same machine):
  - The username is always the logged-in user on the machine.
  - If authorization is enabled, the authorized role is also set to the username.
- For command, toolkit, web user interface, and REST API requests issued to a *remote* integration node or integration server (where *remote* means on a separate machine or using a connection that is specified using a hostname and port):
  - If basic authentication is enabled, the username is set to the authenticated user as verified through a locally-created user using the **mqsiwebuseradmin** command or LDAP.
  - If basic authentication is not enabled, the username is `no-auth-user`.
  - If authorization is enabled, the authorized role is the role that the user is assigned through the **mqsiwebuseradmin** command or LDAP groups. If the user has multiple roles, the authorized role is the one that granted them access for the request URL and method.
  - If authorization is not enabled, the authorized role is always `no-auth-role`.

You can disable the admin log by setting the **enabled** property in the **AdminLog** section of the `node.conf.yaml` or `server.conf.yaml` file for the integration node or server to **false**. The changes take effect when the integration node or server is restarted. For more information about how to configure administration logging for an integration node or server, see “Configuring administration logging” on page 333.

Filtering admin log entries in the web user interface

You can filter the admin log entries that are displayed in the App Connect Enterprise web user interface.

Before you begin

Read the following topics:

- “Viewing administration activity in the admin log” on page 330
- “Configuring administration logging” on page 333

About this task

You can apply filters to control the set of events that are displayed in the web user interface, by entering a search string. You can include one or more tags in the search string, and use exact matching, wildcards, and full regular expressions.

For example, if you want to find all entries that relate to a message flow called `myAsyncRequestResponseFlow`, you can enter the `MSGFLOW:` tag followed by the name of the message flow in the search field:

```
MSGFLOW:myAsyncRequestResponseFlow
```

As a result of this search, the entries are filtered to display only the activities that were carried out against the `myAsyncRequestResponseFlow` message flow.

You can use the `*` wildcard character to represent a tag with any value (a tag must be present but it can have any value). For example, to find out details of the BAR file in which the message flow was deployed,

you can filter the entries to show all events that relate to the specified message flow and that also contain a BAR_FILE: tag:

```
MSGFLOW:myAsyncRequestResponseFlow BAR_FILE:*
```

The resulting filtered list displays only events relating to the myAsyncRequestResponseFlow message flow, including details of the BAR file in which it was deployed.

You can also use regular expression syntax in the search string to set patterns for matching the entries required. For example, to display all the deploy events that have occurred as a result of the **mqsideploy** command being run, you can search for matches to the REQUEST_ID: tags that include mqsideploy, preceded by a regular expression .* to match the timestamp prefix, and followed by .* to match the process ID suffix. By also including the HTTP_METHOD tag that matches POST, only the event that occurs when the command is run is displayed, and not all the events relating to the deployment resource changes that result from the deploy:

```
REQUEST_ID:.*mqsideploy.* HTTP_METHOD:POST
```

Writing admin log entries to a file

Configure an integration node or server to write admin log entries to a file, by setting properties in the node.conf.yaml or server.conf.yaml configuration file.

Before you begin

Read the following topics:

- "Viewing administration activity in the admin log" on page 330
- "Configuring administration logging" on page 333

About this task

In addition to displaying admin log entries in the web user interface, you can configure the integration node or integration server to write the contents of the admin log to a file.

For independent integration servers, the admin log files are created in the adminLog subdirectory of the integration server's working directory:

```
workDir/adminLog
```

For integration nodes and their managed integration servers, the admin log files are created in the common/adminLog directory (*workDir*/common/adminLog), with a subdirectory for each integration node and integration server:

```
workDir/common/adminLog/integrationNodeName/integrationServerName
```

For integration nodes that were created with a shared work path, the admin logs are created in the shared work path to ensure that they are accessible to both nodes in a multi-instance configuration.

Procedure

Configure an integration node or server to write a log of admin events to a file, by modifying the node.conf.yaml or server.conf.yaml configuration file for the integration node or server:

1. Open the appropriate node.conf.yaml or server.conf.yaml configuration file by using a YAML editor.

   If you are not able to access a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters. Tab characters are not valid in YAML files and they will cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

2. Set the following properties in the **AdminLog** section of the appropriate `node.conf.yaml` or `server.conf.yaml` file:
   a) Enable the contents of the admin log to be written to a file, by setting the **fileLog** property to `true`. By default, this property is set to `false`.
   b) Specify the number of days for which the log files are to be stored in the file system before being deleted, by setting the **fileLogRetentionPeriod** property.
   c) Specify the number of files that are to be stored each day, by setting the **fileLogCountDaily** property.
   d) Specify the maximum file log size to be stored each day, by setting the **fileLogSize** property. By default, if file logging is enabled, each daily log file is stored for 30 days before it is deleted, and for each of those 30 days, up to 10 files are stored, up to a maximum of 100MB each per day.
   For example:

```
AdminLog:
  enabled: true               # Control logging admin log messages. Set to true or false,
 default is true.
  # When enabled the maximum amount of disk space required for admin log files is
  # fileLogRetentionPeriod * fileLogCountDaily * fileLogSize
  fileLog: false              # Control writing admin log messages to file. Set to true or
 false, default is false.
  fileLogRetentionPeriod: 30  # Sets the number of days to record admin log.
                              # After this, old files are deleted as new ones are created.
 Default is 30 days.
  fileLogCountDaily: 10       # Maximum number of admin log files to write per day,
 default is 10 per day.
  fileLogSize: 100            # Maximum size in MB for each admin log file. Maximum size
 is 2000MB, default size is 100MB.
  consoleLog: false           # Control writing admin log messages to standard out. Set to
 true or false, default is false.
  consoleLogFormat: 'idText'    # Control the format of admin log messages written to
 standard out. Set to idText, text, or ibmjson. Default is idText.
```

   At the end of the specified retention period, the next log file to be created causes any log files older than the retention period to be deleted. When the maximum number of daily files is exceeded, the oldest file on that day is deleted, and the remaining files are renamed.
3. Restart the integration node or integration server for the changes to take effect. Changes to the `server.conf.yaml` and `node.conf.yaml` files take effect only when the modified integration node or server is restarted.

Configuring administration logging

Configure the logging of administration activity for an integration node or server by setting properties in the `node.conf.yaml` or `server.conf.yaml` configuration file.

Before you begin

Read the topic "Viewing administration activity in the admin log" on page 330.

About this task

The admin log is enabled by default. You can view the events that are held in memory by integration nodes or servers, either in the App Connect Enterprise web user interface or by using the administration REST API.

You can also configure an integration node or server to write the admin log data to a file, as described in "Writing admin log entries to a file" on page 332.

For independent integration servers, you can enable the admin log to write to the App Connect Enterprise console, by setting the **consoleLog** property in the **AdminLog** section of the `server.conf.yaml` file to `true`. By default, this value is set to `false`. You can specify the format of the admin log messages that

are written to standard out by setting the **consoleLogFormat** property to `ibmjson`, `text`, or `idText`. If no value is set, the default format is `idText`. For example:

```
AdminLog:
  enabled: true               # Control logging admin log messages. Set to true or false,
 default is true.
  # When enabled the maximum amount of disk space required for admin log files is
  # fileLogRetentionPeriod * fileLogCountDaily * fileLogSize
  fileLog: false              # Control writing admin log messages to file. Set to true or
 false, default is false.
  fileLogRetentionPeriod: 30  # Sets the number of days to record admin log.
                              # After this, old files are deleted as new ones are created.
 Default is 30 days.
  fileLogCountDaily: 10       # Maximum number of admin log files to write per day, default is
 10 per day.
  fileLogSize: 100            # Maximum size in MB for each admin log file. Maximum size is
 2000MB, default size is 100MB.
  consoleLog: true            # Control writing admin log messages to standard out. Set to true
 or false, default is false.
  consoleLogFormat: 'ibmjson' # Control the format of admin log messages written to standard
 out. Set to idText, text, or ibmjson. Default is idText if unset.
```

You can disable the admin log by setting the *enabled* property to **false**.

The changes to the `server.conf.yaml` or `node.conf.yaml` file take effect only when the integration node or server is restarted.

Managing resources by using the administration REST API

IBM App Connect Enterprise supports the Representational State Transfer Application Programming Interface (REST API), which you can use to administer integration nodes, integration servers, and other resources.

About this task

IBM App Connect Enterprise provides support for the REST API on Linux, AIX, and Windows platforms.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser by specifying the address of your integration node or integration server, and the administration REST API port, followed by `/apidocs`. For example, to display the REST API specification for an integration node, enter a URL as shown in the following example (specifying either `http` or `https`):

```
https://localhost:4414/apidocs
```

where `localhost` is the address of the integration node, and 4414 is the admin REST API port specified in the `node.conf.yaml` file.

To display the REST API specification for an integration server, enter a URL as shown in the following example (specifying either `http` or `https`):

```
https://green.company.com:7600/apidocs
```

where `green.company.com` is the address of the integration server, and 7600 is the admin REST API port specified in the `server.conf.yaml` file.

You can also see details of the REST API here: REST API for administering integration nodes and integration servers where:

- `openapi-appconnectenterprise.yaml` details the REST API for administering integration nodes.
- `openapi-appconnectenterpriseserver.yaml` details the REST API for administering integration servers.

The REST API documentation is also provided with your installation of App Connect Enterprise, in the following directories:

- *install_dir*\server\nodejs\node_modules\@ibm-app-connect\ace-admin-api\docs\server
- *install_dir*\server\nodejs\node_modules\@ibm-app-connect\ace-admin-api\docs\node

where *install_dir* is the directory in which App Connect Enterprise is installed.

For more information about how to use the administration REST API, see the following topics:

- "Administering integration servers by using the administration REST API" on page 335
- "Using trace through the administration REST API" on page 345
- "Monitoring by using the administration REST API" on page 364
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388
- "Using resources statistics through the administration REST API" on page 432
- "Setting message flow user-defined properties at run time by using the administration REST API" on page 436
- "Updating the HTTPS Connector resource manager by using the administration REST API" on page 440

Administering integration servers by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer integration servers.

About this task

You can use the administration REST API to perform the following functions:

- Create, display, start, stop, restart, and delete integration servers that are managed by an integration node.
- Deploy resources to, and delete resources from independent integrations servers and managed integration servers.
- Shut down independent integration servers.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see "Managing resources by using the administration REST API" on page 334.

Displaying integration servers by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the properties of managed integration servers.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering integration servers by using the administration REST API" on page 335

Procedure

- Display the properties of an integration server:

```
GET http://hostname:port/apiv2/servers/integrationServerName
```

For example, to display the properties of integration server ACESERV1, run the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned:


```
{
    "hasChildren": true,
    "name": "ACESERV1",
    "type": "integrationServer",
    "uri": "/apiv2/servers/ACESERV1",
    "properties": {
        "brokerDefaultCCSID": "5348",
        "dataSourceName": "",
        "dataSourcePassword": "********",
        "dataSourceUserId": "LocalSystem",
        "defaultConfigurationTimeout": 300,
        "defaultQueueManager": "",
        "forceServerHTTPS": false,
        "jvmDebugPort": 0,
        "jvmMaxHeapSize": 268435456,
        "jvmMinHeapSize": 33554432,
        "lilPath": "",
        "mqTrustedQueueManager": "no",
        "name": "ACESERV1",
        "stopIfDefaultQMUnavailable": false,
        "trace": "none",
        "traceNodeLevel": true,
        "traceSize": "1G",
        "type": "IntegrationServer"
    },
    "descriptiveProperties": {
        "buildLevel": "ib000-L200318.17215 (S000-L200318.16524)",
        "platformArchitecture": "AMD64",
        "platformName": "Windows 10 Enterprise",
        "platformVersion": "6.3 build 19041 ",
        "productName": "IBM App Connect Enterprise",
        "version": "11.0.0.8"
    },
    "active": {
        "defaultApplicationName": "ACESERV1_DefaultApplication",
        "exceptionLoggingOn": false,
        "isRunning": true,
        "lastMessageEpoch": 0,
        "lastMessageTime": "1970-01-01T00:00:00Z",
        "monitoring": "inactive",
        "monitoringProfile": "",
        "processId": 24096,
        "serviceTraceOn": false,
        "startupEpoch": 1604864621,
        "startupTime": "2020-11-08T19:43:41Z",
        "state": "started",
        "userTraceOn": false
    },

    "links": []
}
```


Additionally, you can use the depth query parameter in the API call to obtain information about more properties of the integration server. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1?depth=2
```

- Display the properties of all managed integration servers:

```
GET http://hostname:port/apiv2/servers
```

For example, run the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

```
{
    "type": "integrationServers",
    "children": [
        {
            "name": "ACESERV1",
            "hasChildren": true,
            "uri": "/apiv2/servers/ACESERV1",
            "type": "integrationServer",
            "active": {
                "processId": 22920,
                "isRunning": true,
                "state": "started"
            }
        },
        {
            "name": "ACESERV2",
            "hasChildren": true,
            "uri": "/apiv2/servers/ACESERV2",
            "type": "integrationServer",
            "active": {
                "processId": 24536,
                "isRunning": true,
                "state": "started"
            }
        }
    ]
}
```

Additionally, you can use the depth query parameter in the API call to obtain information about more properties of the integration server. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers?depth=2
```

Creating an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to create an integration server that is managed by the integration node.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering integration servers by using the administration REST API” on page 335

Procedure

- Create a managed integration server:

```
POST http://hostname:port/apiv2/servers
```

For example, to create an integration server called ACESERV1, run the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers --header 'content-type:application/json' --
data '{"name":"ACESERV1","type":"integrationServer"}'
```

If the command is successful, an HTTP status code 202 is returned.

Starting an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to start a managed integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering integration servers by using the administration REST API" on page 335

Procedure

- Start an integration server:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/start
  ```

  For example, to start the integration server called ACESERV1, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/start
  ```

  If the command is successful, an HTTP status code 202 is returned.

Stopping an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to stop a managed integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering integration servers by using the administration REST API" on page 335

Procedure

- Stop an integration server:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/stop
  ```

  For example, to stop the integration server called ACESERV1, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/stop
  ```

  If the command is successful, an HTTP status code 204 is returned.

Restarting integration servers by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to restart a specific managed integration server or all integration servers that are managed by the integration node.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering integration servers by using the administration REST API" on page 335

Procedure

- Restart an integration server:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/restart
  ```

  For example, to restart the integration server called ACESERV1, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/restart
  ```

  If the command is successful, an HTTP status code 200 is returned.
- Restart all integration servers:

  ```
  POST http://hostname:port/apiv2/restart-all-servers
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/restart-all-servers
  ```

  If the command is successful, an HTTP status code 200 is returned.

Deleting an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to delete a managed integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering integration servers by using the administration REST API” on page 335

Procedure

- Delete an integration server:

  ```
  DELETE http://hostname:port/apiv2/servers/integrationServerName
  ```

  For example, to delete the integration server called ACESERV1, run the following curl command:

  ```
  curl -X DELETE http://hostname:port/apiv2/servers/ACESERV1
  ```

  If the command is successful, an HTTP status code 200 is returned.

Deploying resources to an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to deploy resources such as BAR files to an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering integration servers by using the administration REST API” on page 335

About this task

This topic describes how to deploy resources to an integration server by using the administration REST API. You can use one of two operations in your administration REST API command to deploy resources:

- `delete-all` with one of the following values:
  - `true` to ensure deletion of all resources already deployed to the integration server before deploying your resource.
  - `false` to perform deployment of the resource without deleting any resources that are already deployed to the integration server.
- `deploy-action` with a value of `ImpactAnalysis` to perform an impact analysis of the deployment without actually deploying the resource.

You can specify `delete-all` or `deploy-action`. You cannot specify both in the same administration REST API command.

Procedure

Deploy a resource.

- For an independent integration server:

```
POST http://hostname:port/apiv2/deploy
```

  For example, to deploy a BAR file called `httpreqreply.bar` and delete all resources already deployed to the integration server, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/deploy?delete-all=true" -H 'accept:application/json'
 -H 'content-type:application/octet-stream' --data-binary @/tmp/httpreqreply.bar
```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

```
{
  "type": "responseLog",
  "count": 3,
  "uri": "",
  "LogEntry": [
    {
      "type": "logEntry",
      "message": {
        "number": 9521,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 2,
        "timestamp": 1592141204,
        "threadId": 20730,
        "threadSequenceNumber": 1
      },
      "text": "BIP9521I: Application 'HttpReqReply' has been deleted. ",
      "detailedText": "The resource 'HttpReqReply' of type 'Application' has been deleted. "
    },
    {
      "type": "logEntry",
      "message": {
        "number": 9332,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 3,
        "timestamp": 1592141204,
        "threadId": 20730,
        "threadSequenceNumber": 2
      },
      "text": "BIP9332I: Application 'HttpReqReply' has been created successfully. ",
      "detailedText": "The resource 'HttpReqReply' of type 'Application' has been
 successfully created. "
    },
    {
```

```
      "type": "logEntry",
      "message": {
        "number": 9326,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 1,
        "timestamp": 1592141204,
        "threadId": 20730,
        "threadSequenceNumber": 3
      },
      "text": "BIP9326I: The source 'rest-deploy.bar' has been successfully deployed. ",
      "detailedText": ""
    }
  ]
}
```

To employ the query parameter `ImpactAnalysis` before deploying a BAR file called `httpreqreply.bar`, use the following curl command:

```
curl -X POST   "http://hostname:port/apiv2/deploy?deploy-action=ImpactAnalysis" -H
 'accept:application/json' -H 'content-type:application/octet-stream' --data-binary @/tmp/
httpreqreply.bar
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

```
{
  "type": "responseLog",
  "count": 2,
  "uri": "",
  "LogEntry": [
    {
      "type": "logEntry",
      "message": {
        "number": 2795,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 2,
        "timestamp": 1592148512,
        "threadId": 31842,
        "threadSequenceNumber": 1
      },
      "text": "BIP2795I: Modified Application 'HttpReqReply' has passed validation. ",
      "detailedText": ""
    },
    {
      "type": "logEntry",
      "message": {
        "number": 2781,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 1,
        "timestamp": 1592148512,
        "threadId": 31842,
        "threadSequenceNumber": 2
      },
      "text": "BIP2781I: BAR file 'rest-deploy.bar' has successfully completed analysis. ",
      "detailedText": ""
    }
  ]
}
```

- For a managed integration server:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/deploy
  ```

  For example, to deploy a BAR file called `httpreqreply.bar` to integration server ACESERV1 without deleting any resources that are deployed to the integration server, use the following curl command:

  ```
  curl --request POST --url "http://hostname:port/apiv2/servers/ACESERV1/deploy?delete-
  all=false" --header 'accept:application/json' --header 'content-type:application/octet-
  stream' --data-binary @C:\Toolkit_workspace\ACET11\workspace\BARfiles\httpreqreply.bar
  ```

  If the command is successful, an HTTP status code 200 is returned.

Deleting resources deployed to an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to delete resources that are deployed to an integration server. The resources to be deleted must already be deployed on the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering integration servers by using the administration REST API” on page 335

About this task

The following sections contain instructions for deleting deployed resources on an integration server:

- “Deleting a specific deployed resource on an integration server” on page 342
- “Deleting all deployed resources on an integration server” on page 343

Deleting a specific deployed resource on an integration server

Procedure

Delete a resource

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/delete
  ```

  For example, to delete the deployed application called `HTTPTestApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/delete?names=HTTPTestApplication
  ```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

  ```
  {
      "type": "responseLog",
      "count": 2,
      "uri": "/apiv2/delete",
      "LogEntry": [
          {
              "type": "logEntry",
              "message": {
                  "number": 1062,
                  "severity": 0,
                  "severityCode": "I",
                  "source": "BIPmsgs",
                  "inserts": 2,
  ```

```
                "timestamp": 1604862778,
                "threadId": 21116,
                "threadSequenceNumber": 1
            },
            "text": "BIP1062I: Removing the following objects from integration server
 'KCTEST':  HTTPTestApplication ",
            "detailedText": ""
        },
        {
            "type": "logEntry",
            "message": {
                "number": 9521,
                "severity": 0,
                "severityCode": "I",
                "source": "BIPmsgs",
                "inserts": 2,
                "timestamp": 1604862783,
                "threadId": 21116,
                "threadSequenceNumber": 2
            },
            "text": "BIP9521I: Application 'HTTPTestApplication' has been deleted. ",
            "detailedText": "The resource 'HTTPTestApplication' of type 'Application' has
 been deleted. "
        }
    ]
}
```

To delete more than one resource, specify a colon-delimited set of deployed resource names. For example, to delete the deployed applications called `HTTPTestApplication` and `MonitoringApp`, use following curl command.

```
curl -X POST http://hostname:port/apiv2/delete?names=HTTPTestApplication:MonitoringApp
```

- For a managed integration server:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/delete
  ```

  For example, to delete the application called `HTTPTestApplication`, which is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/delete?names=HTTPTestApplication
  ```

  To delete more than one resource, specify a colon-delimited set of deployed resource names. For example, to delete the applications called `HTTPTestApplication` and `MonitoringApp`, which are deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/delete?
  names=HTTPTestApplication:MonitoringApp
  ```

  To delete a policy project that contains non-dynamic policies, the **"restart-all-applications=true"** query parameter is required. When you specify this parameter, all applications are stopped prior to the resources being deleted, and the applications are then restarted according to the value specified in the **startMode** parameter. Errors are reported for any remaining resources that cannot be started due to missing dependencies.

  For example, to delete the policy project called `theJDBCPolicies`, which is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/delete?
  names=theJDBCPolicies&restart-all-applications=true
  ```

Deleting all deployed resources on an integration server

Procedure

Delete all deployed resources.

- For an independent integration server:

```
POST http://hostname:port/apiv2/delete-all
```

For example, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/delete-all
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned:


```
{
    "type": "responseLog",
    "count": 3,
    "uri": "/apiv2/delete-all",
    "LogEntry": [
        {
            "type": "logEntry",
            "message": {
                "number": 9521,
                "severity": 0,
                "severityCode": "I",
                "source": "BIPmsgs",
                "inserts": 2,
                "timestamp": 1604862622,
                "threadId": 19616,
                "threadSequenceNumber": 1
            },
            "text": "BIP9521I: Application 'MonitoringPayloadApp' has been deleted. ",
            "detailedText": "The resource 'MonitoringPayloadApp' of type 'Application' has
 been deleted. "
        },
        {
            "type": "logEntry",
            "message": {
                "number": 9521,
                "severity": 0,
                "severityCode": "I",
                "source": "BIPmsgs",
                "inserts": 2,
                "timestamp": 1604862622,
                "threadId": 19616,
                "threadSequenceNumber": 2
            },
            "text": "BIP9521I: Application 'MyHttpTestApp' has been deleted. ",
            "detailedText": "The resource 'MyHttpTestApp' of type 'Application' has been
 deleted. "
        },
        {
            "type": "logEntry",
            "message": {
                "number": 9521,
                "severity": 0,
                "severityCode": "I",
                "source": "BIPmsgs",
                "inserts": 2,
                "timestamp": 1604862622,
                "threadId": 19616,
                "threadSequenceNumber": 3
            },
            "text": "BIP9521I: PolicyProject 'MonitoringPolicyProject' has been deleted. ",
            "detailedText": "The resource 'MonitoringPolicyProject' of type 'PolicyProject'
 has been deleted. "
        }
    ]
}
```


- For a managed integration server:

```
POST http://hostname:port/apiv2/servers/integrationServerName/delete-all
```

For example, to delete all resources deployed on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/delete-all
```

Shutting down an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to shut down an independent integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering integration servers by using the administration REST API" on page 335

Procedure

- Shut down an integration server:

  ```
  POST http://hostname:port/apiv2/shutdown
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/shutdown
  ```

  If the command is successful, an HTTP status code 204 is returned.

Using trace through the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer integration nodes, integration servers, and other resources.

About this task

You can set, update, and display trace properties on integration nodes, integration servers, and Trace nodes in integration flows.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see "Managing resources by using the administration REST API" on page 334.

Using trace on an integration node by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to set, update, and display the trace properties of an integration node.

Before you begin

Read "Managing resources by using the administration REST API" on page 334.

Procedure

Use the information in the following topics to help you set, update, and display trace properties for an integration node.

- "Displaying trace properties on an integration node by using the administration REST API" on page 347
- "Setting trace on an integration node by using the administration REST API" on page 346
- "Updating trace properties on an integration node by using the administration REST API" on page 346

Setting trace on an integration node by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to enable trace for an integration node dynamically, without having to restart the integration node.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration node by using the administration REST API" on page 345

About this task

You can use the following REST API method to enable trace on an integration node:

```
POST http://hostname:port/apiv2/trace/agent/service-trace
```

The changes take effect without restarting the integration node.

For example, use the following curl command to enable trace at debug level with trace size of 2048576KB (200MB):

```
curl -X POST -H 'accept:application/json' -H 'content-type:application/json' http://
hostname:port/apiv2/trace/agent/service-trace
-d '{"properties":{"traceLevel":"debug", "traceMode":"safe", "traceSizeKb":2048576}}'
```

The POST operation updates the **Active** section of the trace object. If the command is successful, an HTTP status code `200` is returned.

Updating trace properties on an integration node by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to update the trace properties of an integration node dynamically, without having to restart the integration node.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration node by using the administration REST API" on page 345

About this task

You can use the following REST API method to update and persist the trace properties of an integration node:

```
POST http://hostname:port/apiv2/trace/agent/service-trace
```

For example, use the following curl command to enable trace at debug level with trace size of 2048576KB (200MB). These options are persisted in the `node.conf.yaml` configuration file in the `overrides` directory of the integration node.

```
curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
hostname:port/apiv2/trace/agent/service-trace
-d '{"properties":{"traceLevel":"debug", "traceMode":"safe", "traceSizeKb":2048576}}'
```

The PATCH operation updates the **properties** section of the trace object.

If the command is successful, an HTTP status code `204` is returned. You can see the updated values in the `overrides` directory of the integration node.

Displaying trace properties on an integration node by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the trace properties of an integration node.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration node by using the administration REST API" on page 345

About this task

You can use the following REST API method to query the trace properties of an integration node:

```
GET http://hostname:port/apiv2/trace/agent
GET http://hostname:port/apiv2/trace/agent/service-trace
```

For example, use the following curl command:

```
CURL -X GET http://hostname:port/apiv2/trace/agent
```

A response similar to the following is returned:

```
{
    "name": "agentTrace",
    "type": "trace",
    "children": {
        "serviceTrace": {
            "name": "serviceTrace",
            "type": "serviceTrace",
            "properties": {
                "traceLevel": "none",
                "traceMode": "safe",
                "traceSizeKb": 1048576
            },
            "active": {
                "traceLevel": "none",
                "traceMode": "safe",
                "traceSizeKb": 1048576
            }
        }
    }
}
```

The **properties** section displays the values that are persisted and the **active** section displays the values that are currently in effect.

Using trace on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to set, update, and display the trace properties of an integration server.

Before you begin

Read "Managing resources by using the administration REST API" on page 334.

About this task

You can dynamically update the trace properties of an integration server by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` subdirectory of the integration server's working directory, which means that the updates are not lost if the integration server is restarted.

The following topics contain information about how to use service trace, user trace, and Trace nodes on an integration server, by using the administration REST API:

Procedure

- “Using service trace on an integration server by using the administration REST API” on page 348
- “Using user trace on an integration server by using the administration REST API” on page 354
- “Using Trace nodes on an integration server by using the administration REST API” on page 360
- “Displaying trace properties on an integration server by using the administration REST API” on page 362

Using service trace on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to start, stop, and reset service trace on an integration server, and to set, update, and display service trace properties.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

The following topics contain information about how to use service trace by using the administration REST API.

- “Setting service trace on an integration server by using the administration REST API” on page 348
- “Updating service trace properties on an integration server by using the administration REST API” on page 349
- “Starting service trace on an integration server by using the administration REST API” on page 350
- “Stopping service trace on an integration server by using the administration REST API” on page 352
- “Resetting service trace on an integration server by using the administration REST API” on page 352
- “Displaying service trace properties for an integration server by using the administration REST API” on page 353

Setting service trace on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to set service trace on an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use the following REST API methods to set service trace on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/trace/service-trace
  ```

  For example, use the following curl command:

  ```
  curl -X POST -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/trace/service-trace
  ```

```
-d '{"properties": { "stopOn": "2060", "traceLevel": "debug", "traceMode":
 "temp","traceSizeKb": 1048576 }}'
```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/trace/service-trace
  ```

  For example, to enable debug-level service trace for integration server ACESERV1, with a trace size of `100MB`, a trace mode of `temp`, and automatic termination of the trace on occurrence of a BIP 2060 message, run the following command:

  ```
  curl -X POST -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/servers/ACESERV1/trace/service-trace
  -d '{"properties": { "stopOn": "2060", "traceLevel": "debug", "traceMode":
   "temp","traceSizeKb": 1048576 }}'
  ```

  In both cases, the POST operation updates the **Active** section of the trace object. If the command is successful, an HTTP status code 200 is returned.

  You can control the granularity of the information captured in the service trace, by setting the **`traceLevel`** property to one of the following values:

  – normal
  – debug
  – debugTree
  – diagnostic
  – diagnosticTree

Updating service trace properties on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to update and persist the service trace properties for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

About this task

You can dynamically update service trace properties by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` subdirectory of the integration server's working directory, which means that the updates are not lost if the integration server is stopped.

You can use the following REST API method to update and persist the service trace properties of an integration server:

Procedure

- For an independent integration server:

  ```
  PATCH http://hostname:port/apiv2/trace/service-trace
  ```

  For example, to enable and persist a debug-level service trace with a trace size of `100MB`, a trace mode of `safe`, and automatic termination of the trace on occurrence of a BIP 2060 message, run the following command:

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/trace/service-trace
  ```

```
-d '{"properties": { "stopOn": "2060", "traceLevel": "debug", "traceMode":
 "safe","traceSizeKb": 1048576 }}'
```

- For an integration server that is managed by an integration node:

  ```
  PATCH http://hostname:port/apiv2/servers/integrationServerName/trace/service-trace
  ```

  For example, to enable and persist a debug-level service trace for integration server ACESERV1, with a trace size of `100MB`, a trace mode of `temp`, and automatic termination of the trace on occurrence of a BIP 2060 message, run the following command:

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/servers/ACESERV1/trace/service-trace
  -d '{"properties": { "stopOn": "2060", "traceLevel": "debug", "traceMode":
   "temp","traceSizeKb": 1048576 }}'
  ```

  The PATCH operation updates the **`properties`** section of the trace object. If the command is successful, an HTTP status code 204 is returned. You can see the updated values in the `overrides` subdirectory of the integration server's working directory.

Starting service trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to start service trace at the required level.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

About this task

You can control the granularity of the information captured in the service trace, by setting the **`traceLevel`** property to one of the following values:

- normal
- debug
- debugTree
- diagnostic
- diagnosticTree

Use the following REST API methods to start service trace on an integration server at the specified debug level:

Procedure

- For an independent integration server:
  - To enable service trace at **debug** level:

    ```
    POST http://hostname:port/apiv2/start-service-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-service-trace
    ```

  - To enable service trace at **debugTree** level:

    ```
    POST http://hostname:port/apiv2/start-service-debugtree-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-service-debugtree-trace
    ```

  - To enable service trace at **diagnostic** level:

    ```
    POST http://hostname:port/apiv2/start-service-diagnostic-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-service-diagnostic-trace
    ```

  - To enable service trace at **diagnosticTree** level:

    ```
    POST http://hostname:port/apiv2/start-service-diagnostictree-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-service-diagnostictree-trace
    ```

  - To enable service trace at **normal** level:

    ```
    POST http://hostname:port/apiv2/start-service-normal-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-service-normal-trace
    ```

- For an integration server that is managed by an integration node:

  To enable service trace at **debug** level:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/start-service-trace
  ```

  For example, use the following curl command to enable debug-level service trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/start-service-trace
  ```

  You can also start the service trace at different trace levels, as shown in the following examples:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/start-service-debugtree-trace
  ```

  ```
  POST http://hostname:port /apiv2/servers/integrationServerName/start-service-diagnostic-trace
  ```

  ```
  POST http://hostname:port /apiv2/servers/integrationServerName/start-service-diagnostictree-
  trace
  ```

  ```
  POST http://hostname:port /apiv2/servers/integrationServerName/start-service-normal-trace
  ```

Stopping service trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to stop service trace on an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use the following REST API method to stop service trace (for any trace level) on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/stop-service-trace
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/stop-service-trace
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/stop-service-trace
  ```

  For example, run the following curl command to stop service trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/stop-service-trace
  ```

Resetting service trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to reset server trace properties for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use the following REST API method to reset the service trace on an integration server and remove any previously-captured trace logs.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/reset-service-trace
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/reset-service-trace
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/reset-service-trace
  ```

  For example, use the following curl command to reset service trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/reset-service-trace
  ```

Displaying service trace properties for an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to display service trace properties for an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use one of the following REST API methods to query the service trace properties of an integration server.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/trace/service-trace
  ```

  For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/trace/service-trace
  ```

  A response similar to the following is returned:

  ```
  {
      "hasChildren": false,
      "name": "service-trace",
      "type": "serviceTrace",
      "uri": "/apiv2/trace/service-trace",
      "properties": {
          "identifier": "service-trace",
          "name": "service-trace",
          "stopOn": "",
          "traceLevel": "debug",
          "traceMode": "safe",
          "traceSizeKb": 2048576,
          "type": "serviceTrace"
      },
      "descriptiveProperties": {},
      "active": {
          "stopOn": "",
  ```

```
        "traceLevel": "debug",
        "traceMode": "safe",
        "traceSizeKb": 2048576
    },
    "actions": {},
    "children": {},
    "links": []
}
```

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/trace/service-trace
  ```

  For example, use the following curl command to get information on the current service trace settings for integration server ACESERV1:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/trace/service-trace
  ```

Using user trace on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to start, stop, and reset user trace on an integration server, and to set, update, and display user trace properties.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

To configure user trace with the administration REST API, complete the following steps.

- “Setting user trace on an integration server by using the administration REST API” on page 354
- “Updating user trace properties on an integration server by using the administration REST API” on page 355
- “Starting user trace on an integration server by using the administration REST API” on page 356
- “Stopping user trace on an integration server by using the administration REST API” on page 357
- “Resetting user trace on an integration server by using the administration REST API” on page 358
- “Displaying user trace properties for an integration server by using the administration REST API” on page 358

Setting user trace on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to set user trace on an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use the following REST API methods to set user trace on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/trace/user-trace
  ```

  For example, use the following curl command to enable the user trace at debug level with a trace size of 1GB:

  ```
  curl -X POST http://hostname:port/apiv2/trace/user-trace -H 'accept:application/json' -H
   'content-type:application/json'
  -d "{"properties":{"traceLevel":"debug","traceMode":"safe","traceSizeKb":1048576}}"
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/trace/user-trace
  ```

  For example, use the following curl command to enable debug-level user trace for integration server ACESERV1, with a trace size of 1GB:

  ```
  curl -X POST -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/servers/ACESERV1/trace/user-trace
  -d '{"properties": {"traceLevel": "debug", "traceMode": "safe","traceSizeKb": 1048576 }}'
  ```

  You can control the granularity of the information captured in the user trace, by setting the **`traceLevel`** property to one of the following values:

  – debug
  – debugTree
  – normal
  – none

Updating user trace properties on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to update and persist the user trace properties for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

About this task

You can dynamically update user trace properties by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` subdirectory of the integration server's working directory, which means that the updates are not lost if the integration server is stopped.

You can use the following REST API method to update and persist the user trace properties of an integration server:

Procedure

- For an independent integration server:

  ```
  PATCH http://hostname:port/apiv2/trace/user-trace
  ```

  For example, use the following curl command to enable a debug-level user trace with a trace size of 1GB:

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/trace/user-trace
  ```

```
-d '{"properties": {"traceLevel": "debug", "traceMode": "safe","traceSizeKb": 1048576 }}'
```

- For an integration server that is managed by an integration node:

  ```
  PATCH http://hostname:port/apiv2/servers/integrationServerName/trace/user-trace
  ```

  For example, use the following curl command to enable a debug-level user trace for integration server ACESERV1, with a trace size of 1GB:

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/servers/ACESERV1/trace/user-trace
  -d '{"properties": {"traceLevel": "debug", "traceMode": "safe","traceSizeKb": 1048576 }}'
  ```

  The PATCH operation updates the **properties** section of the trace object. If the command is successful, an HTTP status code 204 is returned. You can see the updated values in the `overrides` subdirectory of the integration server's working directory.

Starting user trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to start user trace at the required level.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

About this task

You can control the granularity of the information captured in the user trace, by setting the **traceLevel** property to one of the following values:

- debug
- debugTree
- normal
- none

You can use the following REST API methods to start user trace on an integration server at the specified debug level, without having to pass the body to the REST API:

Procedure

- For an independent integration server:
  - To enable user trace at **debug** level:

    ```
    POST http://hostname:port/apiv2/start-user-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-user-trace
    ```

  - To enable user trace at **debugTree** level:

    ```
    POST http://hostname:port/apiv2/start-user-debugtree-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-user-debugtree-trace
    ```

  - To enable user trace at **normal** level:

    ```
    POST http://hostname:port/apiv2/start-user-normal-trace
    ```

    For example:

    ```
    curl -X POST http://hostname:port/apiv2/start-user-normal-trace
    ```

- For an integration server that is managed by an integration node:

  To enable user trace at **debug** level:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/start-user-trace
  ```

  For example, use the following curl command to enable debug-level user trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/start-user-trace
  ```

  You can also start the user trace at different trace levels, as shown in the following examples:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/start-user-debugtree-trace
  ```

  ```
  POST http://hostname:port /apiv2/servers/integrationServerName/start-user-normal-trace
  ```

Stopping user trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to stop user trace on an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration server by using the administration REST API" on page 347

Procedure

You can use the following REST API method to stop user trace (for any trace level) on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/stop-user-trace
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/stop-user-trace
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/stop-user-trace
  ```

  For example, run the following curl command to stop user trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/stop-user-trace
  ```

Resetting user trace on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to reset user trace properties for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration server by using the administration REST API" on page 347

Procedure

You can use the following REST API method to reset the user trace on an integration server and remove any previously captured trace logs.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/reset-user-trace
  ```

  For example, run the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/reset-user-trace
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/reset-user-trace
  ```

  For example, use the following curl command to reset user trace for integration server ACESERV1:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/reset-user-trace
  ```

Displaying user trace properties for an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to display user trace properties for an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration server by using the administration REST API" on page 347

Procedure

You can use one of the following REST API methods to query the user trace properties of an integration server.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/trace/user-trace
  ```

  For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/trace/user-trace
  ```

  A response similar to the following is returned:

  ```
  {
      "hasChildren": false,
      "name": "user-trace",
      "type": "userTrace",
      "uri": "/apiv2/trace/user-trace",
      "properties": {
          "name": "",
          "traceLevel": "none",
          "traceMode": "safe",
          "traceSizeKb": 1048576,
          "type": ""
      },
      "descriptiveProperties": {},
      "active": {
          "traceLevel": "none",
          "traceMode": "safe",
          "traceSizeKb": 1048576
      },
      "actions": {},
      "children": {},
      "links": []
  }
  ```

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/trace/user-trace
  ```

  For example, use the following curl command to get user trace details for integration server ACESERV1:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/trace/user-trace
  ```

  A response similar to the following is returned:

  ```
  {
      "hasChildren": false,
      "name": "user-trace",
      "type": "userTrace",
      "uri": "/apiv2/servers/ACESERV1/trace/user-trace",
      "properties": {
          "name": "",
          "traceLevel": "none",
          "traceMode": "safe",
          "traceSizeKb": 1048576,
          "type": ""
      },
      "descriptiveProperties": {},
      "active": {
          "traceLevel": "none",
          "traceMode": "safe",
          "traceSizeKb": 1048576
      },
      "actions": {},
      "children": {},
      "links": []
  }
  ```

Using Trace nodes on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to enable and disable Trace nodes and to display the Trace node properties of an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration server by using the administration REST API" on page 347

Procedure

The following topics contain information about enabling, disabling, and displaying properties for Trace nodes.

- "Enabling and disabling Trace nodes on an integration server by using the administration REST API" on page 360
- "Displaying Trace node properties on an integration server by using the administration REST API" on page 361

Enabling and disabling Trace nodes on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to enable or disable Trace nodes on an integration server, without having to restart the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using trace on an integration server by using the administration REST API" on page 347

Procedure

You can use the following REST API methods to set service trace on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/trace/trace-nodes
  ```

  For example, use the following curl command to disable Trace nodes on the integration server:

  ```
  curl -X POST  -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/trace/trace-nodes
  -d '{ "properties": {"enabled": false  } }'
  ```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/trace/trace-nodes
  ```

  For example, use the following curl command to disable Trace nodes in integration server ACESERV1:

  ```
  curl -X POST  -H 'accept:application/json' -H 'content-type:application/json'  http://
  hostname:port/apiv2/servers/ACESERV1/trace/trace-nodes
  -d '{ "properties": {"enabled": false  } }'
  ```

  The POST operation updates the **Active** section of the traceNode object. If the command is successful, an HTTP status code 200 is returned.

You can persist the Trace node options by using the PATCH API:

- For an independent integration server:

```
PATCH http://hostname:port /apiv2/trace/trace-nodes
```

  For example, use the following curl command to disable Trace nodes and persist the setting in the integration server's `server.conf.yaml` configuration file:

```
curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
hostname:port/apiv2/trace/trace-nodes
-d '{ "properties": {"enabled": false  } }'
```

- For an integration server that is managed by an integration node:

```
PATCH http://hostname:port /apiv2/servers/integrationServerName/trace/trace-nodes
```

  For example, use the following curl command to disable Trace nodes in integration server ACESERV1 and persist the setting in the `server.conf.yaml` file:

```
curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
hostname:port/apiv2/servers/ACESERV1/trace/trace-nodes
-d '{ "properties": {"enabled": false  } }'
```

  The PATCH operation updates the **`properties`** section of the traceNode object. If the command is successful, an HTTP status code 204 is returned. You can see the updated values in the `overrides` subdirectory of the integration server's working directory.

Displaying Trace node properties on an integration server by using the administration REST API
You can use the IBM App Connect Enterprise administration REST API to display the Trace node properties of an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

You can use the following REST API method to view the Trace node properties of an integration server.

- For an independent integration server:

```
GET http://hostname:port/apiv2/trace/trace-nodes
```

  For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/trace/trace-nodes
```

  A response similar to the following is returned:

```
{
    "hasChildren": false,
    "name": "trace-nodes",
    "type": "traceNodes",
    "uri": "/apiv2/trace/trace-nodes",
    "properties": {
        "enabled": true,
        "identifier": "user-trace",
        "name": "user-trace",
        "type": "userTrace"
    },
    "descriptiveProperties": {},
    "active": {
        "enabled": true
    },
    "actions": {},
    "children": {},
```

```
    "links": []
}
```

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/trace/trace-nodes
  ```

  For example, use the following curl command to get user trace details of integration server ACESERV1:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/trace/trace-nodes
  ```

  A response similar to the following is returned:

  ```
  {
      "hasChildren": false,
      "name": "trace-nodes",
      "type": "traceNodes",
      "uri": "/apiv2/servers/ACESERV1/trace/trace-nodes",
      "properties": {
          "enabled": false,
          "identifier": "user-trace",
          "name": "user-trace",
          "type": "userTrace"
      },
      "descriptiveProperties": {},
      "active": {
          "enabled": true
      },
      "actions": {},
      "children": {},
      "links": []
  }
  ```

Displaying trace properties on an integration server by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display trace properties for an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using trace on an integration server by using the administration REST API” on page 347

Procedure

Use one of the following REST API methods to query the trace properties of an integration server.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/trace
  ```

  For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/trace
  ```

  A response similar to the following is returned:

  ```
  {
      "hasChildren": true,
      "name": "trace",
      "type": "trace",
      "uri": "/apiv2/trace",
      "properties": {
          "name": "",
          "type": ""
      },
      "descriptiveProperties": {},
  ```

```
    "active": {},
    "actions": {},
    "children": {
        "userTrace": {
            "hasChildren": false,
            "name": "user-trace",
            "type": "userTrace",
            "uri": "/apiv2/trace/user-trace"
        },
        "serviceTrace": {
            "hasChildren": false,
            "name": "service-trace",
            "type": "serviceTrace",
            "uri": "/apiv2/trace/service-trace"
        },
        "traceNodes": {
            "hasChildren": false,
            "name": "trace-nodes",
            "type": "traceNodes",
            "uri": "/apiv2/trace/trace-nodes"
        }
    },
    "links": []
}
```

- For an integration server that is managed by an integration node:

```
GET http://hostname:port/apiv2/servers/integrationServerName/trace
```

For example, use the following curl command to query the trace properties of integration server ACESERV1:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1/trace
```

A response similar to the following is returned:

```
{
    "hasChildren": true,
    "name": "trace",
    "type": "trace",
    "uri": "/apiv2/servers/ACESERV1/trace",
    "properties": {
        "name": "",
        "type": ""
    },
    "descriptiveProperties": {},
    "active": {},
    "actions": {},
    "children": {
        "userTrace": {
            "hasChildren": false,
            "name": "user-trace",
            "type": "userTrace",
            "uri": "/apiv2/servers/ACESERV1/trace/user-trace"
        },
        "serviceTrace": {
            "hasChildren": false,
            "name": "service-trace",
            "type": "serviceTrace",
            "uri": "/apiv2/servers/ACESERV1/trace/service-trace"
        },
        "traceNodes": {
            "hasChildren": false,
            "name": "trace-nodes",
            "type": "traceNodes",
            "uri": "/apiv2/servers/ACESERV1/trace/trace-nodes"
        }
    },
    "links": []
}
```

Monitoring by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer the monitoring of applications, message flows, and integration solutions that have been developed by using the REST API or integration services.

About this task

You can use the administration REST API to start and stop monitoring, and to extract, attach, and detach monitoring profiles.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see “Managing resources by using the administration REST API” on page 334.

Activating monitoring by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to activate the monitoring of applications, message flows, and integration solutions that have been developed by using the REST API or integration services.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Monitoring by using the administration REST API” on page 364

About this task

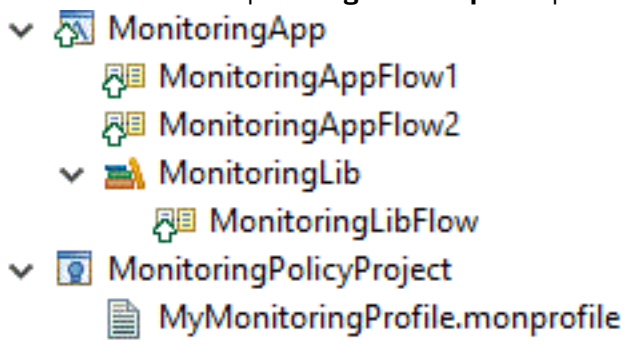
This topic describes:

- “Activating monitoring on an application” on page 365
- “Activating monitoring on a REST API integration solution” on page 366
- “Activating monitoring on an integration services integration solution ” on page 367
- “Activating monitoring on all integration solutions” on page 368

The examples in this topic are based on the following resources:

- An application: `MonitoringApp`
- Message flows packaged with the application: `MonitoringAppFlow1` and `MonitoringAppFlow2`
- A library associated with the application: `Monitoring Lib`
- A message flow in the library: `MonitoringLibFlow`

as shown in this example **Integration Explorer** pane excerpt:



Activating monitoring on an application

About this task

You can activate monitoring on an application and the message flows that are packaged in it, or you can activate monitoring on a single message flow:

Procedure

- Activate monitoring on an application and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/start-monitoring
```

  For example, to activate monitoring on the application `MonitoringApp`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/start-monitoring
```

  For example, to activate monitoring on the application `MonitoringApp` on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/start-
monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

- Activate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageflowName/
start-monitoring
```

  For example, to activate monitoring on the message flow `MonitoringAppFlow1`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/messageflows/
MonitoringAppFlow1/start-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
messageflows/messageflowName/start-monitoring
```

  For example, to activate monitoring on the message flow `MonitoringLibFlow`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/libraries/
MonitoringLib/messageflows/MonitoringLibFlow/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/messageflows/messageflowName/start-monitoring
```

  For example, to activate monitoring on the message flow `MonitoringAppFlow1` that is deployed on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
messageflows/MonitoringAppFlow1/start-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/libraries/libraryName/messageflows/messageflowName/start-
monitoring
```

  For example, to activate monitoring on the message flow `MonitoringLibFlow` that is deployed on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
libraries/MonitoringLib/messageflows/MonitoringLibFlow/start-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the application and message flows.

Activating monitoring on a REST API integration solution

About this task

You can activate monitoring on an integration solution, including the message that are packaged in it, that has been developed by using a REST API or you can activate monitoring on a single message flow in such an integration solution:

Procedure

- Activate monitoring on a REST API integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
start-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.
- Activate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageflowName/start-
monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageflowName/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageflowName/start-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageflowName/start-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the application and message flows.

Activating monitoring on an integration services integration solution

About this task

You can activate monitoring on an integration solution, including the message that are packaged in it, that has been developed by using integration services or you can activate monitoring on a single message flow in such an integration solution:

Procedure

- Activate monitoring on an integration services integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/start-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

- Activate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/intergrationServiceName/
messageflows/messageflowName/start-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/services/intergrationServiceName/libraries/libraryName/
messageflows/messageflowName/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/intergationServiceName/messageflows/messageflowName/start-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/intergationServiceName/libraries/libraryName/messageflows/messageflowName/start-
monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the integration solution and message flows.

Activating monitoring on all integration solutions

About this task

You can use a single command in the administration REST API to activate monitoring on all integration solutions and message flows that have been developed in applications, including those that have been developed by using a REST API or integrations services, and all those in libraries.

Procedure

- Activate monitoring on all integration solutions and their message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/monitoring/flow-monitoring/start-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2 servers/integrationServerName/monitoring/flow-monitoring/
start-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the integration solutions and message flows.

Deactivating monitoring by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to deactivate the monitoring of applications, message flows, and integration solutions that have been developed by using the REST API or integration services.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Monitoring by using the administration REST API” on page 364

About this task

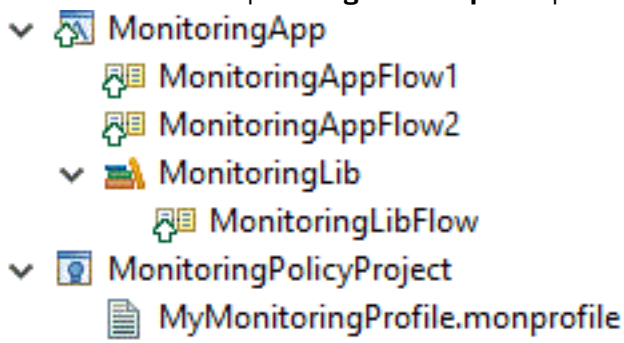
This topic describes:

- “Deactivating monitoring on an application” on page 369
- “Deactivating monitoring on a REST API integration solution” on page 371
- “Deactivating monitoring on an integration services integration solution ” on page 372
- “Deactivating monitoring on all integration solutions” on page 373

The examples in this topic are based on the following resources:

- An application: `MonitoringApp`
- Message flows packaged with the application: `MonitoringAppFlow1` and `MonitoringAppFlow2`
- A library associated with the application: `Monitoring Lib`
- A message flow in the library: `MonitoringLibFlow`

as shown in this example **Integration Explorer** pane excerpt:



Deactivating monitoring on an application

About this task

You can deactivate monitoring on an application and the message flows that are packaged in it, or you can deactivate monitoring on a single message flow:

Procedure

- Deactivate monitoring on an application and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/stop-monitoring
```

  For example, to deactivate monitoring on the application MonitoringApp, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/stop-monitoring
```

  For example, to deactivate monitoring on the application MonitoringApp on integration server ACESERV1 , use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/stop-
monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

- Deactivate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageflowName/
stop-monitoring
```

  For example, to deactivate monitoring on the message flow `MonitoringAppFlow1`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/messageflows/
MonitoringAppFlow1/stop-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
messageflows/messageflowName/stop-monitoring
```

  For example, to deactivate monitoring on the message flow `MonitoringLibFlow`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/libraries/
MonitoringLib/messageflows/MonitoringLibFlow/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/messageflows/messageflowName/stop-monitoring
```

  For example, to deactivate monitoring on the message flow `MonitoringAppFlow1` that is deployed on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
messageflows/MonitoringAppFlow1/stop-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/libraries/libraryName/messageflows/messageflowName/stop-
monitoring
```

  For example, to deactivate monitoring on a message flow called `MonitoringLibFlow` on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
libraries/MonitoringLib/messageflows/MonitoringLibFlow/stop-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the application and message flows.

Deactivating monitoring on a REST API integration solution

About this task

You can deactivate monitoring on an integration solution, including the message that are packaged in it, that has been developed by using a REST API or you can deactivate monitoring on a single message flow in such an integration solution:

Procedure

- Deactivate monitoring on a REST API integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/stop-
monitoring
```

  If the command is successful, an HTTP status code 200 is returned.
- Deactivate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageflowName/stop-
monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageflowName/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageflowName/stop-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageflowName/stop-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the application and message flows.

Deactivating monitoring on an integration services integration solution

About this task

You can use deactivate monitoring on an integration solution, including the message that are packaged in it, that has been developed by using integration services or you can deactivate monitoring on a single message flow in such an integration solution:

Procedure

- Deactivate monitoring on an integration services integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/stop-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

- Deactivate monitoring on a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/intergrationServiceName/
messageflows/messageflowName/stop-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/services/intergrationServiceName/libraries/libraryName/
messageflows/messageflowName/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/intergationServiceName/messageflows/messageflowName/stop-monitoring
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/intergationServiceName/libraries/libraryName/messageflows/messageflowName/stop-
monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the integration solution and message flows.

Deactivating monitoring on all integration solutions

About this task

You can use a single command in the administration REST API to deactivate monitoring on all integration solutions and message flows that have been developed in applications, including those that have been developed by using a REST API or integrations services, and all those in libraries.

Procedure

- Deactivate monitoring on all integration solutions and their message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/monitoring/flow-monitoring/stop-monitoring
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2 servers/integrationServerName/monitoring/flow-monitoring/
stop-monitoring
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify the monitoring status of the integration solutions and message flows.

Extracting a monitoring profile by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to extract monitoring profiles of message flows that are running in applications and in integration solutions that have been developed by using the REST API or integration services.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Monitoring by using the administration REST API” on page 364

About this task

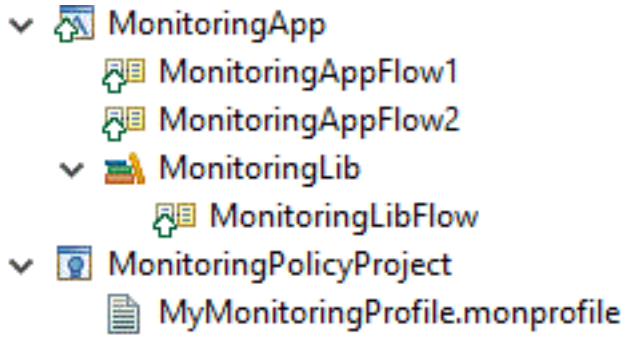
This topic describes the following scenarios.

- “Extracting the monitoring profile of a message flow in an application ” on page 374
- “Extracting the monitoring profile of a message flow in a REST API integration solution” on page 377
- “Extracting the monitoring profile of a message flow in an integration services integration solution” on page 377

The examples in this topic are based on the following resources:

- An application: `MonitoringApp`
- Message flows packaged with the application: `MonitoringAppFlow1` and `MonitoringAppFlow2`
- A library associated with the application: `Monitoring Lib`
- A message flow in the library: `MonitoringLibFlow`

as shown in this example **Integration Explorer** pane excerpt:



Extracting the monitoring profile of a message flow in an application

Procedure

You can extract the monitoring profile of a message flow in an application.

- For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageflowName/
extract-monitoring-profile
```

For example, to extract the monitoring profile of the message flow `MonitoringAppFlow1`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/messageflows/
MonitoringAppFlow1/extract-monitoring-profile
```

A response similar to the following is returned:

```
{
    "MonitoringProfile": {
        "HTTP Input.transaction.Start": {
            "bitstreamData": [
                {
                    "Content": "none",
                    "Encoding": "none"
                }
            ],
            "eventFilter": [
                {
                    "Text": "true()",
                    "isXPath": true
                }
            ],
            "eventName": [
                {
                    "Text": "HTTP Input.TransactionStart",
                    "isXPath": false
                }
            ],
            "eventType": "Transaction start"
        }
    }
}
```

If the message flow is in a library:

```
POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
messageflows/messageflowName/extract-monitoring-profile
```

For example, to extract the monitoring profile of the message flow `MonitoringLibFlow`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/libraries/MonitoringLib/
messageflows/MonitoringLibFlow/extract-monitoring-profile
```

A response similar to the following is returned:

```
{
    "MonitoringProfile": {
        "HTTP Input.transaction.Start": {
            "bitstreamData": [
                {
                    "Content": "none",
                    "Encoding": "none"
                }
            ],
            "eventFilter": [
                {
                    "Text": "true()",
                    "isXPath": true
                }
            ],
            "eventName": [
                {
                    "Text": "HTTP Input.TransactionStart",
                    "isXPath": false
                }
            ],
            "eventType": "Transaction start"
```

```
        },
        "HTTP Input.transaction.End": {
            "bitstreamData": [
                {
                    "Content": "none",
                    "Encoding": "none"
                }
            ],
            "eventFilter": [
                {
                    "Text": "true()",
                    "isXPath": true
                }
            ],
            "eventName": [
                {
                    "Text": "HTTP Input.TransactionEnd",
                    "isXPath": false
                }
            ],
            "eventType": "Transaction end"
        },
        "HTTP Reply.terminal.in": {
            "bitstreamData": [
                {
                    "Content": "none",
                    "Encoding": "none"
                }
            ],
            "eventFilter": [
                {
                    "Text": "true()",
                    "isXPath": true
                }
            ],
            "eventName": [
                {
                    "Text": "HTTP Reply.InTerminal",
                    "isXPath": false
                }
            ],
            "eventType": "In terminal"
        }
    }
}
```

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  messageflows/messageflowName/extract-monitoring-profile
  ```

  For example, to extract the monitoring profile of the message flow `MonitoringAppFlow1` that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
  messageflows/MonitoringAppFlow1/extract-monitoring-profile
  ```

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  libraries/libraryName/messageflows/messageflowName/extract-monitoring-profile
  ```

  For example, to extract the monitoring profile of the message flow `MonitoringLibFlow` that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
  libraries/MonitoringLib/messageflows/MonitoringLibFlow/extract-monitoring-profile
  ```

  If the command is successful, an HTTP status code 200 is returned.

Extracting the monitoring profile of a message flow in a REST API integration solution

Procedure

You can extract the monitoring profile of a message flow in a REST API integration solution.

- For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageflowName/extract-
monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageflowName/extract-monitoring-profile
```

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageflowName/extract-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageflowName/extract-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

Extracting the monitoring profile of a message flow in an integration services integration solution

Procedure

You can extract the monitoring profile of a message flow in an integration services integration solution.

- For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/messageflowName/
extract-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
messageflows/messageflowName/extract-monitoring-profile
```

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/messageflows/messageflowName/extract-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/libraries/libraryName/messageflows/messageflowName/extract-
monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

Attaching a monitoring profile by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to attach a monitoring profile to an application and the message flows that are packaged with it. You can also attach a monitoring profile to an individual message flow that is deployed in an application or static library.

Before you begin

Read the following topics:

- "Monitoring basics" on page 2772
- "Managing resources by using the administration REST API" on page 334
- "Monitoring by using the administration REST API" on page 364

Follow the instructions in "Creating a monitoring profile" on page 2793 to learn how to create a monitoring profile.

After you have created the monitoring profile, save it in a policy project and deploy the policy project in either an independent BAR file or in the same BAR file as an associated application or message flow. Attach the monitoring profile to an application or message flow. You can then activate monitoring by using the administration REST API as described in "Activating monitoring by using the administration REST API" on page 364.

About this task

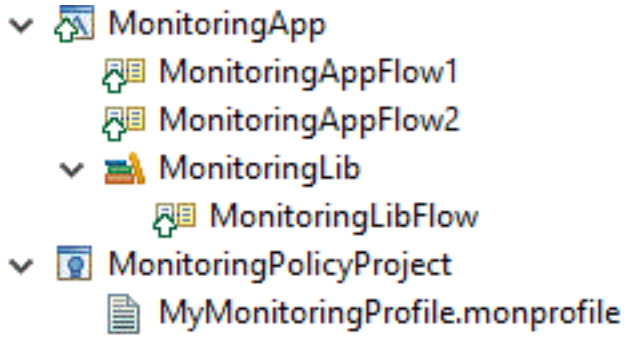
This topic describes:

- "Attaching a monitoring profile to an application" on page 378
- "Attaching a monitoring profile to a REST API integration solution" on page 381
- "Attaching a monitoring profile to an integration services integration solution " on page 381

The examples in this topic are based on the following resources:

- An application: `MonitoringApp`
- Message flows packaged with the application: `MonitoringAppFlow1` and `MonitoringAppFlow2`
- A library associated with the application: `Monitoring Lib`
- A message flow in the library: `MonitoringLibFlow`
- A policy project: `MonitoringPolicyProject`
- A monitoring profile in the policy project: `MyMonitoringProfile`

as shown in this example **Integration Explorer** pane excerpt:



Attaching a monitoring profile to an application

About this task

You can attach a monitoring profile to an application and it message flows, or you can attach a monitoring profile to a single message flow:

Procedure

- Attach a monitoring profile to an application and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/attach-monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the application `MonitoringApp`, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/applications/MonitoringApp/attach-monitoring-
profile?profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/attach-monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the application `MonitoringApp` on integration server ACESERV1, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
attach-monitoring-profile?profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  If the command is successful, an HTTP status code 200 is returned.

- Attach a monitoring profile to a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageflowName/
attach-monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the message flow `MonitoringAppFlow1`, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/applications/MonitoringApp/
messageflows/MonitoringAppFlow1/attach-monitoring-profile?
profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
messageflows/messageflowName/attach-monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the message flow `MonitoringLibFlow`, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/applications/MonitoringApp/libraries/
MonitoringLib/messageflows/MonitoringLibFlow/attach-monitoring-profile?
profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/messageflows/messageflowName/attach-monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the message flow `MonitoringAppFlow1` on integration server ACESERV1, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/servers/ACESERV1/applications/
MonitoringApp/messageflows/MonitoringAppFlow1/attach-monitoring-profile?
profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/libraries/libraryName/messageflows/messageflowName/attach-
monitoring-profile
```

  For example, to attach the monitoring profile `MyMonitoringProfile` to the message flow `MonitoringLibFlow` on integration server ACESERV1, use the following curl command:

```
curl -X POST "http://hostname:port/apiv2/servers/ACESERV1/applications/
MonitoringApp/messageflows/MonitoringLibFlow/attach-monitoring-profile?
profile=MyMonitoringProfile&project=MonitoringPolicyProject"
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been attached to the application or message flow.

Attaching a monitoring profile to a REST API integration solution

Procedure

- Attach a monitoring profile to a REST API integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/attach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
attach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.
- Attach a monitoring profile to a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageflowName/
attach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageflowName/attach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageflowName/attach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageflowName/attach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been attached to the application or message flow.

Attaching a monitoring profile to an integration services integration solution

Procedure

- Attach a monitoring profile to an integration service integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/attach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/attach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

- Attach a monitoring profile to a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/services/intergrationServiceName/
messageflows/messageflowName/attach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
messageflows/messageflowName/attach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/intergationServiceName/messageflows/messageflowName/attach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/libraries/libraryName/messageflows/messageflowName/attach-
monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been attached to the application or message flow.

Detaching a monitoring profile by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to detach a monitoring profile from an application and the message flows that are packaged with it. You can also detach a monitoring profile from an individual message flow that is deployed in an application or static library.

Before you begin

Read the following topics:

- “Monitoring basics” on page 2772
- “Managing resources by using the administration REST API” on page 334
- “Monitoring by using the administration REST API” on page 364

About this task

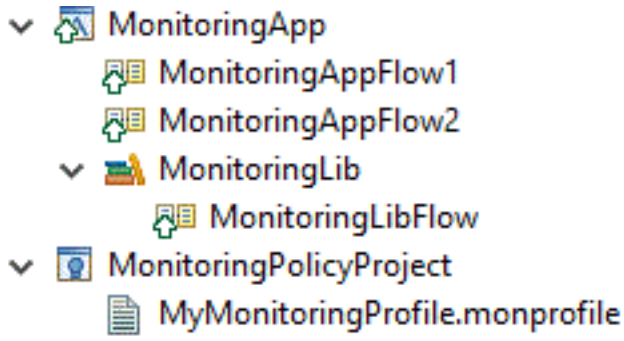
This topic describes:

- “Detaching a monitoring profile from an application” on page 383
- “Detaching a monitoring profile from a REST API integration solution” on page 385
- “Detaching a monitoring profile from an integration services integration solution ” on page 385

The examples in this topic are based on the following resources:

- An application: `MonitoringApp`
- Message flows packaged with the application: `MonitoringAppFlow1` and `MonitoringAppFlow2`
- A library associated with the application: `Monitoring Lib`
- A message flow in the library: `MonitoringLibFlow`

as shown in this example **Integration Explorer** pane excerpt:



Detaching a monitoring profile from an application

About this task

You can detach a monitoring profile from an application or from a single message flow:

Procedure

- Detach a monitoring profile from an application and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/detach-monitoring-profile
```

  For example, to detach the monitoring profile from the application `MonitoringApp`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/detach-monitoring-
profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/servers/integrationServerName/apiv2/
applications/applicationName/detach-monitoring-profile
```

  For example, to detach the monitoring profile from the application `MonitoringApp` on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
detach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

- Detach a monitoring profile from a single message flow:
  a) For an independent integration server:

```
 POST http://hostname:port/apiv2/applications/applicationName/
messageflows/messageflowName/detach-monitoring-profile
```

  For example, to detach the monitoring profile from the message flow `MonitoringAppFlow1`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/messageflows/
MonitoringAppFlow1/detach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
messageflows/messageflowName/detach-monitoring-profile
```

  For example, to detach the monitoring profile from the message flow `MonitoringLibFlow`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/libraries/
MonitoringLib/messageflows/MonitoringLibFlow/detach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
messageflows/messageflowName/detach-monitoring-profile
```

  For example, to detach the monitoring profile from the message flow `MonitoringAppFlow1` on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
messageflows/MonitoringAppFlow1/detach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/libraries/libraryName/messageflows/messageflowName/detach-
monitoring-profile
```

  For example, to detach the monitoring profile from the message flow `MonitoringLibFlow` on integration server ACESERV1, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/MonitoringApp/
libraries/MonitoringLib/messageflows/MonitoringLibFlow/detach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been detached from the application or message flow.

Detaching a monitoring profile from a REST API integration solution

Procedure

- Detach a monitoring profile to a REST API integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/detach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
detach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.
- Detach a monitoring profile from a single message flow:
  a) For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageflowName/
detach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageflowName/detach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageflowName/detach-monitoring-profile
```

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageflowName/detach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been detached from the application or message flow.

Detaching a monitoring profile from an integration services integration solution

Procedure

- Detach a monitoring profile from an integration service integration solution and its message flows:
  a) For an independent integration server:

```
POST http://hostname:port /apiv2/services/integrationServiceName/detach-monitoring-profile
```

  b) For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/detach-monitoring-profile
```

  If the command is successful, an HTTP status code 200 is returned.

- Detach a monitoring profile from a single message flow:

  a) For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/intergrationServiceName/
  messageflows/messageflowName/detach-monitoring-profile
  ```

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/services/intergrationServiceName/libraries/libraryName/
  messageflows/messageflowName/detach-monitoring-profile
  ```

  b) For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/intergationServiceName/messageflows/messageflowName/detach-monitoring-profile
  ```

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/intergrationServiceName/libraries/libraryName/messageflows/messageflowName/
  detach-monitoring-profile
  ```

  If the command is successful, an HTTP status code 200 is returned.

What to do next

You can use the **mqsireportflowmonitoring** command to verify that the monitoring profile has been detached from the application or message flow.

Displaying monitoring status by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the monitoring status for an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Monitoring by using the administration REST API" on page 364

Procedure

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/monitoring/flow-monitoring
  ```

  For example, run the following curl command:

  ```
  curl -X GET http://localhost:port/apiv2/monitoring/flow-monitoring
  ```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

  ```
  {
      "hasChildren": false,
      "name": "flow-monitoring",
      "type": "flowMonitoring",
      "uri": "/apiv2/monitoring/flow-monitoring",
      "properties": {
          "eventFormat": "MonitoringEventV2",
          "name": "MessageFlow",
          "publicationOn": "active",
          "type": "MessageFlow",
          "useParserNameInPayload": false
      },
  ```

```
    "descriptiveProperties": {},
    "active": {
        "publicationOn": "active"
    },
    "actions": {
        "unavailable": {
            "start-monitoring": "/apiv2/monitoring/flow-monitoring/start-monitoring"
        },
        "available": {
            "stop-monitoring": "/apiv2/monitoring/flow-monitoring/stop-monitoring"
        }
    },
    "children": {},
    "links": []
}
```

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/monitoring/flow-monitoring
  ```

  For example, to display the monitoring status for integration server ACESERV1, run the following command:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/monitoring/flow-monitoring
  ```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

  ```
  {
      "hasChildren": false,
      "name": "flow-monitoring",
      "type": "flowMonitoring",
      "uri": "/apiv2/servers/ACESERV1/monitoring/flow-monitoring",
      "properties": {
          "eventFormat": "MonitoringEventV2",
          "name": "MessageFlow",
          "publicationOn": "active",
          "type": "MessageFlow",
          "useParserNameInPayload": false
      },
      "descriptiveProperties": {},
      "active": {
          "publicationOn": "active"
      },
      "actions": {
          "unavailable": {
              "start-monitoring": "/apiv2/servers/ACESERV1/monitoring/flow-monitoring/start-
  monitoring"
          },
          "available": {
              "stop-monitoring": "/apiv2/servers/ACESERV1/monitoring/flow-monitoring/stop-
  monitoring"
          }
      },
      "children": {},
      "links": []
  }
  ```

Updating monitoring status by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to update and persist the monitoring status for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Monitoring by using the administration REST API” on page 364

About this task

You can dynamically update monitoring status by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` subdirectory of the integration server's working directory, which means that the updates are not lost if the integration server is stopped.

You can use the following REST API method to update and persist the monitoring status of an integration server:

Procedure

- For an independent integration server:

  ```
  PATCH http://hostname:port/apiv2/monitoring/flow-monitoring
  ```

  For example, to set monitoring status to `inactive` and persist this status, run the curl following command:

  ```
  curl -X PATCH http://hostname:port/apiv2/monitoring/flow-monitoring -d {"properties":
   { "publicationOn": "active" }}
  ```

- For an integration server that is managed by an integration node:

  ```
  PATCH http://hostname:port/apiv2/servers/integrationServerName/monitoring/flow-monitoring
  ```

  For example, to set monitoring status to `inactive` and persist this status on integration server ACESERV1, run the following curl command:

  ```
  curl -X PATCH http://hostname:port/apiv2/servers/ACESERV1/monitoring/flow-monitoring -d
   {"properties": { "publicationOn": "active" }}
  ```

  The PATCH operation updates the **publicationOn** setting in the Monitoring object. If the command is successful, an HTTP status code 204 is returned. You can see the updated values in the `overrides` subdirectory of the integration server's working directory.

Administering applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer applications, REST APIS, and integration services that have been deployed to an integration server.

About this task

Use the administration REST API to perform the following tasks:

- Display the applications, REST APIs, and integration services that are deployed in an integration server.
- Start applications, REST APIs, and integration services.
- Stop applications, REST APIs, and integration services.
- Tear down applications, REST APIs, and integration services.
- Delete applications, REST APIs, and integration services.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see “Managing resources by using the administration REST API” on page 334.

Displaying applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the applications, REST APIs, and integration services that are deployed in an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388

About this task

This topic describes:

- "Displaying applications in an integration server" on page 389
- "Displaying REST APIs in an integration server" on page 391
- "Displaying integration services in an integration server " on page 391

Displaying applications in an integration server

Procedure

Display the applications.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/applications
  ```

  For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/applications
  ```

  If the command is successful and there are applications deployed in the integration server, an HTTP status code 200 and a response similar to the following are returned. In this example, there are 2 applications deployed in the integration server: `HTTPInputApplication` and `Transformation_Map`:

  ```
  {
    "hasChildren": true,
    "name": "applications",
    "type": "applications",
    "uri": "/apiv2/ applications",
    "properties": {},
    "descriptiveProperties": {},
    "active": {},
    "actions": {},
    "children": [
      {
        "hasChildren": true,
        "name": "HTTPInputApplication",
        "type": "application",
        "uri": "/apiv2/applications/HTTPInputApplication"
      },
      {
        "hasChildren": true,
        "name": "Transformation_Map",
        "type": "application",
        "uri": "/apiv2/ applications/Transformation_Map"
      }
    ],
    "links": []
  ```

```
}
```

If the command is successful but there are no applications deployed in the integration server, an HTTP status code 200 and a response similar to the following are returned:

```
{
  "hasChildren": false,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [],
  "links": []
}
```

Additionally, you can use the depth query parameter in the API call to obtain more information about the applications. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/applications?depth=2
```

For an example of the response that is returned for such an API call, see “Display applications on an independent integration server” on page 402.

- For an integration server that is managed by an integration node:

```
GET http://hostname:port/apiv2/servers/integrationServerName/applications
```

For example, to display the applications that are deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1/applications
```

If the command is successful and there are applications deployed in the integration server, an HTTP status code 200 and a response similar to the following are returned. In this example, there are 2 applications deployed in the integration server: `HTTPInputApplication` and `Transformation_Map`:

```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication"
    },
    {
      "hasChildren": true,
      "name": "Transformation_Map",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map"
    }
  ],
  "links": []
}
```

If the command is successful but there are no applications deployed in the integration server, an HTTP status code 200 and a response similar to the following are returned:

```
{
  "hasChildren": false,
  "name": "applications",
```

```
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [],
  "links": []
}
```

Additionally, you can use the depth query parameter in the API call to obtain more information about the applications. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/integrationServerName/applications?depth=2
```

For an example of the response that is returned for such an API call, see "Display applications on an integration server that is managed by an integration node" on page 404.

Displaying REST APIs in an integration server

Procedure

Display the REST APIs.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/rest-apis
  ```

  If the command is successful, an HTTP status code 200 and information about the REST APIS that are deployed, if any, in the integration server are returned.

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/rest-apis
  ```

  If the command is successful, an HTTP status code 200 and information about the REST APIS that are deployed, if any, in the integration server are returned.

Displaying integration services in an integration server

Procedure

Display the integration services.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/services
  ```

  If the command is successful, an HTTP status code 200 and information about the integration services that are deployed, if any, in the integration server are returned.

- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/services
  ```

  If the command is successful, an HTTP status code 200 and information about the integration services that are deployed, if any, in the integration server are returned.

Starting applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to start named applications, REST APIs, and integration services that are deployed on an integration server. The application, REST API, or integration service to be started must already be deployed in the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388

About this task

This topic describes:

- "Starting an application on an integration server" on page 392
- "Starting a REST API on an integration server" on page 393
- "Starting an integration service on an integration server " on page 393

Starting an application on an integration server

Procedure

Start an application.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/start
  ```

  For example, to start the application that is called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the application has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the application has started. For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that an application has started on an independent integration server" on page 407.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  start
  ```

  For example, to start the application that is called HTTPInputApplication and deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the application has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the application has started. For example, to display detailed information about applications on integration server ACESERV1, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that an application has started on an integration server that is managed by an integration node" on page 409.

Starting a REST API on an integration server

Procedure

Start a REST API.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the REST API has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the REST API has started.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the REST API has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the REST API has started.

Starting an integration service on an integration server

Procedure

Start an integration service.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the integration service has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the integration service has started.
- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/start
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the integration service has started, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the integration service has started.

Stopping applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to stop named applications, REST APIs, and integration services that are deployed on an integration server. The application, REST API, or integration service to be stopped must already be deployed on the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388

About this task

This topic describes:

- "Stopping an application on an integration server" on page 394
- "Stopping a REST API on an integration server" on page 395
- "Stopping an integration service on an integration server " on page 396

Stopping an application on an integration server

Procedure

Stop an application.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/stop
  ```

  For example, to stop the application that is called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the application has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the application has stopped. For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that an application has stopped on an independent integration server" on page 410.
- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  stop
  ```

  For example, to stop the application that is called `HTTPInputApplication` deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the application has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the application has stopped. For example, to display the information about applications on integration server ACESERV1, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/servers/ACESERV1/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that an application has stopped on an integration server that is managed by an integration node" on page 412.

Stopping a REST API on an integration server

Procedure

Stop a REST API.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the REST API has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the REST API has stopped.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the REST API has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the REST API has stopped.

Stopping an integration service on an integration server

Procedure

Stop an integration service.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the integration service has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the integration service has stopped.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/stop
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the integration service has stopped, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the detailed information that shows whether the integration service has stopped.

Tearing down applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to tear down named applications, REST APIs, and integration services that are deployed on an integration server. The application, REST API, or integration service to be torn down must already be deployed on the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388

About this task

The action of tearing down terminates all connections and threads that are associated with the application, REST API, or integration service. The application, REST API, or integration service remains in the runtime image of the integration server and can be restarted by using a start action without the need to redeploy. There is no need to stop and restart the integration server to restart the application, REST API, or integration service.

This topic describes:

Tearing down an application on an integration server

Procedure

Tear down an application.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/teardown
  ```

  For example, to tear down the application that is called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful. For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that a teardown action has completed on an independent integration server" on page 414.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  teardown
  ```

  For example, to tear down the application that is called `HTTPInputApplication` deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful. For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2servers/ACESERV1/applications?depth=2
  ```

  For an example of the response that is returned for such an API call, see "Check that a teardown action has completed on an integration server that is managed by an integration node" on page 415.

Tearing down a REST API on an integration server

Procedure

Tear down a REST API.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful.

Tearing down an integration service on an integration server

Procedure

Tear down an integration service.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/teardown
  ```

  If the command is successful, an HTTP status code 200 is returned. No response message is returned.

  To verify that the tear down request has been successful, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the request was successful.

  ```
  curl -X GET http://hostname:port/apiv2/servers/integrationServerName/services?depth=2
  ```

Deleting applications, REST APIs, and integration services by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to delete named applications, REST APIs, and integration services that are deployed on an integration server. The application, REST API, or integration service to be deleted must already be deployed on the integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Administering applications, REST APIs, and integration services by using the administration REST API" on page 388

About this task

The action of deleting an application, REST API, or integration service includes issuing a tear down request followed by the deletion. The teardown operation is part of the deletion operation: you do not have to issue a teardown request. The application, REST API, or integration service is deleted from the integration server and from the configuration on disk; the application, REST API, or integration service is undeployed.

This topic describes:

- "Deleting an application on an integration server" on page 399
- "Deleting a REST API on an integration server" on page 401
- "Deleting an integration service on an integration server " on page 401

Deleting an application on an integration server

Procedure

Delete an application.

- For an independent integration server:

  ```
  DELETE http://hostname:port/apiv2/applications/applicationName
  ```

  For example, to delete the application that is called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X DELETE http://hostname:port/apiv2/applications/HTTPInputApplication
  ```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

  ```
  {
    "type": "responseLog",
    "count": 1,
    "uri": "",
    "LogEntry": [
      {
        "type": "logEntry",
        "message": {
          "number": 9521,
          "severity": 0,
          "severityCode": "I",
          "source": "BIPmsgs",
          "inserts": 2,
          "timestamp": 1592387071,
          "threadId": 4460,
          "threadSequenceNumber": 1
        },
        "text": "BIP9521I: Application 'HTTPInputApplication' has been deleted. ",
  ```

```
      "detailedText": "The resource 'HTTPInputApplication' of type 'Application' has been
 deleted. "
    }
  ]
}
```

Additionally, to verify that the application has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the application has been deleted. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/applications?depth=2
```

For an example of the response that is returned for such an API call, see "Check that an application has been deleted from an independent integration server" on page 416.

- For an integration server that is managed by an integration node:

```
DELETE http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName
```

For example, to delete the application that is called `HTTPInputApplication` deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X DELETE http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

```
{
  "type": "responseLog",
  "count": 1,
  "uri": "",
  "LogEntry": [
    {
      "type": "logEntry",
      "message": {
        "number": 9521,
        "severity": 0,
        "severityCode": "I",
        "source": "BIPmsgs",
        "inserts": 2,
        "timestamp": 1591203977,
        "threadId": 4460,
        "threadSequenceNumber": 1
      },
      "text": "BIP9521I: Application 'HTTPInputApplication' has been deleted. ",
      "detailedText": "The resource 'HTTPInputApplication' of type 'Application' has been
 deleted. "
    }
  ]
}
```

Additionally, to verify that the application has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the application has been deleted. For example, to display the information about applications on integration server ACESERV1, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1/applications?depth=2
```

For an example of the response that is returned for such an API call, see "Check that an application has been deleted from an integration server that is managed by an integration node" on page 417.

Deleting a REST API on an integration server

Procedure

Delete a REST API.

- For an independent integration server:

  ```
  DELETE http://hostname:port/apiv2/rest-apis/rest-apiName
  ```

  If the command is successful, an HTTP status code 200 is returned.

  To verify that the REST API has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the REST API has been deleted.

- For an integration server that is managed by an integration node:

  ```
  DELETE http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName
  ```

  If the command is successful, an HTTP status code 200 is returned.

  To verify that the REST API has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the REST API has been deleted.

Deleting an integration service on an integration server

Procedure

Delete an integration service.

- For an independent integration server:

  ```
  DELETE http://hostname:port/apiv2/services/integrationServiceName
  ```

  If the command is successful, an HTTP status code 200 is returned.

  To verify that the integration service has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the integration service has been deleted.

- For an integration server that is managed by an integration node:

  ```
  DELETE http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName
  ```

  If the command is successful, an HTTP status code 200 is returned.

  To verify that the integration service has been deleted, use the GET API call as described in "Displaying applications, REST APIs, and integration services by using the administration REST API" on page 389. Specify a value of 2 on the depth query parameter in the API call to obtain the information that shows whether the integration service has been deleted.

Example responses to the GET API administration REST API call with a value of 2 on the depth query parameter

Using a GET API administration REST API call with a value of 2 on the depth query parameter, you can display useful information about the status of your applications.

This topic provides example responses to a GET API administration REST API call with a value of 2 on the depth query parameter.

For example, use the following curl command to display information about applications on an independent integration server:

```
curl -X GET http://hostname:port/apiv2/applications?depth=2
```

To display the information about applications on integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X GET  http://hostname:port/apiv2/servers/ACESERV1/applications?depth=2
```

The following sections show example responses.

- "Display applications" on page 402
- "Check that an application has started" on page 407
- "Check that an application has stopped" on page 410
- "Check that a teardown action has completed" on page 414
- "Check that an application has been deleted" on page 416

Display applications

Display applications on an independent integration server

In this case, two applications named HTTPInputApplication and Transformation_Map are deployed.


```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": true,
        "javaIsolation": true,
        "monitoring": "inactive",
```

```
        "monitoringProfile": "",
        "resourceState": "stateStarted",
        "startupEpoch": 1591027127,
        "startupTime": "2020-06-01T15:58:47Z",
        "state": "started"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/detach-
monitoring-profile",
          "setup": "/apiv2/applications/HTTPInputApplication/setup",
          "start": "/apiv2/ applications/HTTPInputApplication/start",
          "stop-monitoring": "/apiv2/applications/HTTPInputApplication/stop-monitoring",
          "validate": "/apiv2/applications/HTTPInputApplication/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/attach-
monitoring-profile",
          "invalidate-security-cache": "/apiv2/applications/HTTPInputApplication/invalidate-
security-cache",
          "start-monitoring": "/apiv2/applications/HTTPInputApplication/start-monitoring",
          "stop": "/apiv2/applications/HTTPInputApplication/stop",
          "teardown": "/apiv2/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    },
    {
      "hasChildren": true,
      "name": "Transformation_Map",
      "type": "application",
      "uri": "/apiv2/applications/Transformation_Map",
      "properties": {
        "identifier": "Transformation_Map",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "Transformation_Map",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2018-03-27 21:46:32",
        "locationOnDisk": "Transformation_Map",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": true,
```

```
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateStarted",
        "startupEpoch": 1591027192,
        "startupTime": "2020-06-01T15:59:52Z",
        "state": "started"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/applications/Transformation_Map/detach-
monitoring-profile",
          "setup": "/apiv2/applications/Transformation_Map/setup",
          "start": "/apiv2/applications/Transformation_Map/start",
          "stop-monitoring": "/apiv2/applications/Transformation_Map/stop-monitoring",
          "validate": "/apiv2/applications/Transformation_Map/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/applications/Transformation_Map/attach-
monitoring-profile",
          "invalidate-security-cache": "/apiv2/applications/Transformation_Map/invalidate-
security-cache",
          "start-monitoring": "/apiv2/applications/Transformation_Map/start-monitoring",
          "stop": "/apiv2/applications/Transformation_Map/stop",
          "teardown": "/apiv2/applications/Transformation_Map/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/applications/Transformation_Map/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/applications/Transformation_Map/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/applications/Transformation_Map/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/applications/Transformation_Map/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/applications/Transformation_Map/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```


Display applications on an integration server that is managed by an integration node

In this case, two applications named HTTPInputApplication and Transformation_Map are deployed on integration server ACESERV1, which is managed by an integration node.


```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
```

```
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": true,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateStarted",
        "startupEpoch": 1591027127,
        "startupTime": "2020-06-01T15:58:47Z",
        "state": "started"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/detach-monitoring-profile",
          "setup": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/setup",
          "start": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start",
          "stop-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop-
monitoring",
          "validate": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/attach-monitoring-profile",
          "invalidate-security-cache": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/invalidate-security-cache",
          "start-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start-
monitoring",
          "stop": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop",
          "teardown": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
```

```
          "type": "statistics",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    },
    {
      "hasChildren": true,
      "name": "Transformation_Map",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map",
      "properties": {
        "identifier": "Transformation_Map",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "Transformation_Map",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2018-03-27 21:46:32",
        "locationOnDisk": "Transformation_Map",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": true,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateStarted",
        "startupEpoch": 1591027192,
        "startupTime": "2020-06-01T15:59:52Z",
        "state": "started"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/Transformation_Map/
detach-monitoring-profile",
          "setup": "/apiv2/servers/ACESERV1/applications/Transformation_Map/setup",
          "start": "/apiv2/servers/ACESERV1/applications/Transformation_Map/start",
          "stop-monitoring": "/apiv2/servers/ACESERV1/applications/Transformation_Map/stop-
monitoring",
          "validate": "/apiv2/servers/ACESERV1/applications/Transformation_Map/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/Transformation_Map/
attach-monitoring-profile",
          "invalidate-security-cache": "/apiv2/servers/ACESERV1/applications/Transformation_Map/
invalidate-security-cache",
          "start-monitoring": "/apiv2/servers/ACESERV1/applications/Transformation_Map/start-
monitoring",
          "stop": "/apiv2/servers/ACESERV1/applications/Transformation_Map/stop",
          "teardown": "/apiv2/servers/ACESERV1/applications/Transformation_Map/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map/messageflows"
```

```
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/servers/ACESERV1/applications/Transformation_Map/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```

Check that an application has started

Check that an application has started on an independent integration server

The application for which information is to be returned is `HTTPInputApplication`. Note the `"started"` state of the application that is indicated in the `"active"` section in the JSON response.

```
{
  "hasChildren": true,
  "name": "HTTPInputApplication",
  "type": "application",
  "uri": "/apiv2/applications/HTTPInputApplication",
  "properties": {
    "identifier": "HTTPInputApplication",
    "javaIsolation": true,
    "monitoring": "inherit",
    "monitoringProfile": "",
    "name": "HTTPInputApplication",
    "startMode": "Maintained",
    "type": "Application"
  },
  "descriptiveProperties": {
    "lastModified": "2015-03-10 14:22:34",
    "locationOnDisk": "HTTPInputApplication",
    "longDesc": "",
    "shortDesc": ""
  },
  "active": {
    "isDefaultApplication": false,
    "isRunning": true,
    "javaIsolation": true,
    "monitoring": "inactive",
    "monitoringProfile": "",
    "resourceState": "stateStarted",
    "startupEpoch": 1591719945,
    "startupTime": "2020-06-09T16:25:45Z",
    "state": "started"
  },
  "actions": {
    "unavailable": {
      "detach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/detach-monitoring-
profile",
      "setup": "/apiv2/applications/HTTPInputApplication/setup",
      "start": "/apiv2/applications/HTTPInputApplication/start",
      "stop-monitoring": "/apiv2/applications/HTTPInputApplication/stop-monitoring",
      "validate": "/apiv2/applications/HTTPInputApplication/validate"
    },
    "available": {
      "attach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/attach-monitoring-
profile",
      "invalidate-security-cache": "/apiv2/applications/HTTPInputApplication/invalidate-
security-cache",
      "start-monitoring": "/apiv2/applications/HTTPInputApplication/start-monitoring",
      "stop": "/apiv2/applications/HTTPInputApplication/stop",
      "teardown": "/apiv2/applications/HTTPInputApplication/teardown"
    }
  },
  "children": {
    "subFlows": {
      "hasChildren": false,
      "name": "subflows",
      "type": "subFlows",
      "uri": "/apiv2/applications/HTTPInputApplication/subflows",
      "properties": {},
```

```
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [],
      "links": []
    },
    "resources": {
      "hasChildren": true,
      "name": "resources",
      "type": "resources",
      "uri": "/apiv2/applications/HTTPInputApplication/resources",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [
        {
          "hasChildren": false,
          "name": "HTTPInputMessageFlow_inputMessage.xml",
          "type": "resource",
          "uri": "/apiv2/applications/HTTPInputApplication/resources/
HTTPInputMessageFlow_inputMessage.xml"
        },
        {
          "hasChildren": false,
          "name": "HTTPInputMessageFlow_Mapping.map",
          "type": "resource",
          "uri": "/apiv2/applications/HTTPInputApplication/resources/
HTTPInputMessageFlow_Mapping.map"
        }
      ],
      "links": []
    },
    "libraries": {
      "hasChildren": false,
      "name": "libraries",
      "type": "libraries",
      "uri": "/apiv2/applications/HTTPInputApplication/libraries",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [],
      "links": []
    },
    "messageFlows": {
      "hasChildren": true,
      "name": "messageflows",
      "type": "messageFlows",
      "uri": "/apiv2/applications/HTTPInputApplication/messageflows",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {
        "available": {
          "start": "/apiv2/applications/HTTPInputApplication/messageflows/start",
          "stop": "/apiv2/applications/HTTPInputApplication/messageflows/stop"
        }
      },
      "children": [
        {
          "hasChildren": true,
          "name": "HTTPInputMessageFlow",
          "type": "messageFlow",
          "uri": "/apiv2/applications/HTTPInputApplication/messageflows/HTTPInputMessageFlow"
        }
      ],
      "links": []
    },
    "statistics": {
      "hasChildren": true,
      "name": "statistics",
      "type": "statistics",
      "uri": "/apiv2/applications/HTTPInputApplication/statistics",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": {
        "snapshot": {
          "hasChildren": false,
          "name": "snapshot",
```

```
          "type": "snapshot",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics/snapshot"
        },
        "archive": {
          "hasChildren": false,
          "name": "archive",
          "type": "archive",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics/archive"
        }
      },
      "links": []
    }
  },
  "links": []
}
```

Check that an application has started on an integration server that is managed by an integration node

The application for which information is to be returned is HTTPInputApplication on integration server ACESERV1, which is managed by an integration node. Note the "started" state of the application that is indicated in the "active" section in the JSON response.

```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": true,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateStarted",
        "startupEpoch": 1591028554,
        "startupTime": "2020-06-01T16:22:34Z",
        "state": "started"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/detach-monitoring-profile",
          "setup": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/setup",
          "start": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start",
          "stop-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop-
monitoring",
          "validate": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/attach-monitoring-profile",
```

```
          "invalidate-security-cache": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/invalidate-security-cache",
          "start-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start-
monitoring",
          "stop": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop",
          "teardown": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```


Check that an application has stopped

Check that an application has stopped on an independent integration server

The application for which information is to be returned is HTTPInputApplication. Note the "stopped" state of the application that is indicated in the "active" section in the JSON response.


```
{
  "hasChildren": true,
  "name": "HTTPInputApplication",
  "type": "application",
  "uri": "/apiv2/applications/HTTPInputApplication",
  "properties": {
    "identifier": "HTTPInputApplication",
    "javaIsolation": true,
    "monitoring": "inherit",
    "monitoringProfile": "",
    "name": "HTTPInputApplication",
    "startMode": "Maintained",
    "type": "Application"
  },
  "descriptiveProperties": {
    "lastModified": "2015-03-10 14:22:34",
    "locationOnDisk": "HTTPInputApplication",
    "longDesc": "",
    "shortDesc": ""
  },
  "active": {
    "isDefaultApplication": false,
    "isRunning": false,
    "javaIsolation": true,
    "monitoring": "inactive",
```

```
    "monitoringProfile": "",
    "resourceState": "stateSetup",
    "startupEpoch": 1591719945,
    "startupTime": "2020-06-09T16:25:45Z",
    "state": "stopped"
  },
  "actions": {
    "unavailable": {
      "detach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/detach-monitoring-
profile",
      "setup": "/apiv2/applications/HTTPInputApplication/setup",
      "stop": "/apiv2/applications/HTTPInputApplication/stop",
      "stop-monitoring": "/apiv2/applications/HTTPInputApplication/stop-monitoring",
      "validate": "/apiv2/applications/HTTPInputApplication/validate"
    },
    "available": {
      "attach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/attach-monitoring-
profile",
      "invalidate-security-cache": "/apiv2/applications/HTTPInputApplication/invalidate-
security-cache",
      "start": "/apiv2/applications/HTTPInputApplication/start",
      "start-monitoring": "/apiv2/applications/HTTPInputApplication/start-monitoring",
      "teardown": "/apiv2/applications/HTTPInputApplication/teardown"
    }
  },
  "children": {
    "subFlows": {
      "hasChildren": false,
      "name": "subflows",
      "type": "subFlows",
      "uri": "/apiv2/applications/HTTPInputApplication/subflows",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [],
      "links": []
    },
    "resources": {
      "hasChildren": true,
      "name": "resources",
      "type": "resources",
      "uri": "/apiv2/applications/HTTPInputApplication/resources",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [
        {
          "hasChildren": false,
          "name": "HTTPInputMessageFlow_inputMessage.xml",
          "type": "resource",
          "uri": "/apiv2/applications/HTTPInputApplication/resources/
HTTPInputMessageFlow_inputMessage.xml"
        },
        {
          "hasChildren": false,
          "name": "HTTPInputMessageFlow_Mapping.map",
          "type": "resource",
          "uri": "/apiv2/applications/HTTPInputApplication/resources/
HTTPInputMessageFlow_Mapping.map"
        }
      ],
      "links": []
    },
    "libraries": {
      "hasChildren": false,
      "name": "libraries",
      "type": "libraries",
      "uri": "/apiv2/applications/HTTPInputApplication/libraries",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": [],
      "links": []
    },
    "messageFlows": {
      "hasChildren": true,
      "name": "messageflows",
      "type": "messageFlows",
      "uri": "/apiv2/applications/HTTPInputApplication/messageflows",
```

```
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {
        "available": {
          "start": "/apiv2/applications/HTTPInputApplication/messageflows/start",
          "stop": "/apiv2/applications/HTTPInputApplication/messageflows/stop"
        }
      },
      "children": [
        {
          "hasChildren": true,
          "name": "HTTPInputMessageFlow",
          "type": "messageFlow",
          "uri": "/apiv2/applications/HTTPInputApplication/messageflows/HTTPInputMessageFlow"
        }
      ],
      "links": []
    },
    "statistics": {
      "hasChildren": true,
      "name": "statistics",
      "type": "statistics",
      "uri": "/apiv2/applications/HTTPInputApplication/statistics",
      "properties": {},
      "descriptiveProperties": {},
      "active": {},
      "actions": {},
      "children": {
        "snapshot": {
          "hasChildren": false,
          "name": "snapshot",
          "type": "snapshot",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics/snapshot"
        },
        "archive": {
          "hasChildren": false,
          "name": "archive",
          "type": "archive",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics/archive"
        }
      },
      "links": []
    }
  },
  "links": []
}
```

Check that an application has stopped on an integration server that is managed by an integration node

The application for which information is to be returned is HTTPInputApplication on integration server ACESERV1, which is managed by an integration node. Note the "stopped" state of the application that is indicated in the "active" section in the JSON response.

```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
```

```
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": false,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateSetup",
        "startupEpoch": 1591200777,
        "startupTime": "2020-06-03T16:12:57Z",
        "state": "stopped"
      },
      "actions": {
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/detach-monitoring-profile",
          "setup": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/setup",
          "stop": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop",
          "stop-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop-
monitoring",
          "validate": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/validate"
        },
        "available": {
          "attach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/attach-monitoring-profile",
          "invalidate-security-cache": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/invalidate-security-cache",
          "start": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start",
          "start-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start-
monitoring",
          "teardown": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```

Check that a teardown action has completed on an independent integration server

A teardown request was issued for the application named HTTPInputApplication. Note the "stopped" value in "state" and the "stateInitialized" value in "resourceState" that are indicated in the "active" section in the JSON response.


```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": false,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateInitialized",
        "startupEpoch": 1591203215,
        "startupTime": "2020-06-03T17:53:35Z",
        "state": "stopped"
      },
      "actions": {
        "available": {
          "attach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/attach-
monitoring-profile",
          "invalidate-security-cache": "/apiv2/applications/HTTPInputApplication/invalidate-
security-cache",
          "setup": "/apiv2/applications/HTTPInputApplication/setup",
          "start": "/apiv2/applications/HTTPInputApplication/start",
          "start-monitoring": "/apiv2/applications/HTTPInputApplication/start-monitoring",
          "validate": "/apiv2/applications/HTTPInputApplication/validate"
        },
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/applications/HTTPInputApplication/detach-
monitoring-profile",
          "stop": "/apiv2/applications/HTTPInputApplication/stop",
          "stop-monitoring": "/apiv2/applications/HTTPInputApplication/stop-monitoring",
          "teardown": "/apiv2/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
```

```
          "type": "resources",
          "uri": "/apiv2/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```


Check that a teardown action has completed on an integration server that is managed by an integration node

A teardown request was issued for the application named HTTPInputApplication on integration server ACESERV1, which is managed by an integration node. Note the "stopped" value in "state" and the "stateInitialized" value in "resourceState" that are indicated in the "active" section in the JSON response.


```
{
  "hasChildren": true,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [
    {
      "hasChildren": true,
      "name": "HTTPInputApplication",
      "type": "application",
      "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication",
      "properties": {
        "identifier": "HTTPInputApplication",
        "javaIsolation": true,
        "monitoring": "inherit",
        "monitoringProfile": "",
        "name": "HTTPInputApplication",
        "startMode": "Maintained",
        "type": "Application"
      },
      "descriptiveProperties": {
        "lastModified": "2015-03-10 14:22:34",
        "locationOnDisk": "HTTPInputApplication",
        "longDesc": "",
        "shortDesc": ""
      },
      "active": {
        "isDefaultApplication": false,
        "isRunning": false,
        "javaIsolation": true,
        "monitoring": "inactive",
        "monitoringProfile": "",
        "resourceState": "stateInitialized",
        "startupEpoch": 1591203215,
        "startupTime": "2020-06-03T16:53:35Z",
        "state": "stopped"
      },
```

```
      "actions": {
        "available": {
          "attach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/attach-monitoring-profile",
          "invalidate-security-cache": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/invalidate-security-cache",
          "setup": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/setup",
          "start": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start",
          "start-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/start-
monitoring",
          "validate": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/validate"
        },
        "unavailable": {
          "detach-monitoring-profile": "/apiv2/servers/ACESERV1/applications/
HTTPInputApplication/detach-monitoring-profile",
          "stop": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop",
          "stop-monitoring": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/stop-
monitoring",
          "teardown": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/teardown"
        }
      },
      "children": {
        "subFlows": {
          "hasChildren": false,
          "name": "subflows",
          "type": "subFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/subflows"
        },
        "resources": {
          "hasChildren": true,
          "name": "resources",
          "type": "resources",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/resources"
        },
        "libraries": {
          "hasChildren": false,
          "name": "libraries",
          "type": "libraries",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/libraries"
        },
        "messageFlows": {
          "hasChildren": true,
          "name": "messageflows",
          "type": "messageFlows",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows"
        },
        "statistics": {
          "hasChildren": true,
          "name": "statistics",
          "type": "statistics",
          "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/statistics"
        }
      },
      "links": []
    }
  ],
  "links": []
}
```


Check that an application has been deleted

Check that an application has been deleted from an independent integration server

If the application that was deleted was the only application to be deployed in the integration server, the response from running the command is similar to the following example. If other applications had also been deployed at the time of the command, they are listed in the response.


```
{
  "hasChildren": false,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [],
```

```
  "links": []
}
```

Check that an application has been deleted from an integration server that is managed by an integration node

If the application that was deleted was the only application to be deployed in the integration server, ACESERV1, which is managed by an integration node, the response from running the command is similar to the following example. If other applications had also been deployed at the time of the command, they are listed in the response.

```
{
  "hasChildren": false,
  "name": "applications",
  "type": "applications",
  "uri": "/apiv2/servers/ACESERV1/applications",
  "properties": {},
  "descriptiveProperties": {},
  "active": {},
  "actions": {},
  "children": [],
  "links": []
}
```

Administering message flows by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer message flows that are deployed to an integration server in applications, REST API projects, and integration service projects.

About this task

Use the administration REST API to perform the following tasks:

- Display message flows that are deployed to an integration server.
- Start message flows.
- Stop message flows.
- Tear down message flows.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see “Managing resources by using the administration REST API” on page 334.

Displaying message flows by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the message flows that are deployed to an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering message flows by using the administration REST API” on page 417

About this task

This topic describes:

- “Displaying message flows in an application” on page 418
- “Displaying message flows in a REST API project” on page 419
- “Displaying message flows in an integration service project ” on page 420

Displaying message flows in an application

About this task

You can display the message flows that are deployed to an integration server in an application.

Procedure

Display the message flows.

- For an independent integration server:

```
GET http://hostname:port/apiv2/applications/applicationName/messageflows
```

For example, to display the message flows in application HTTPInputApplication, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows
```

If the command is successful and there are message flows deployed to the integration server in application HTTPInputApplication, an HTTP status code 200 and a response similar to the following are returned:

```
{
    "hasChildren": true,
    "name": "messageflows",
    "type": "messageFlows",
    "uri": "/apiv2/applications/HTTPInputApplication/messageflows",
    "properties": {},
    "descriptiveProperties": {},
    "active": {},
    "actions": {
        "available": {
            "start": "/apiv2/applications/HTTPInputApplication/messageflows/start",
            "stop": "/apiv2/applications/HTTPInputApplication/messageflows/stop"
        }
    },
    "children": [
        {
            "hasChildren": true,
            "name": "RequestService",
            "type": "messageFlow",
            "uri": "/apiv2/applications/HTTPInputApplication/messageflows/RequestService"
        },
        {
            "hasChildren": true,
            "name": "ResponseService",
            "type": "messageFlow",
            "uri": "/apiv2/applications/HTTPInputApplication/messageflows/ResponseService"
        }
    ],
    "links": []
}
```

Additionally, you can use the depth query parameter in the API call to obtain more information about the message flows. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/applications/applicationName/messageflows?depth=2
```

- For an integration server that is managed by an integration node:

```
GET http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
messageflows
```

  For example, to display the message flows that are deployed in application HTTPInputApplication to integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
messageflows
```

  If the command is successful and there are message flows deployed to the integration server, an HTTP status code 200 and a response similar to the following are returned:

```
{
    "hasChildren": true,
    "name": "messageflows",
    "type": "messageFlows",
    "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows",
    "properties": {},
    "descriptiveProperties": {},
    "active": {},
    "actions": {
        "available": {
            "start": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows/
start",
            "stop": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows/
stop"
        }
    },
    "children": [
        {
            "hasChildren": true,
            "name": "RequestService",
            "type": "messageFlow",
            "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows/
RequestService"
        },
        {
            "hasChildren": true,
            "name": "ResponseService",
            "type": "messageFlow",
            "uri": "/apiv2/servers/ACESERV1/applications/HTTPInputApplication/messageflows/
ResponseService"
        }
    ],
    "links": []
}
```

  Additionally, you can use the depth query parameter in the API call to obtain more information about the message flows. For example, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/integrationServerName/
applications/applicationName/messageflows?depth=2
```

Displaying message flows in a REST API project

About this task

You can display the message flows that are deployed to an integration server in a REST API project.

Procedure

Display the message flows.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows
  ```

  If the command is successful, an HTTP status code 200 and information about the message flows that are deployed in the REST API project, if any, are returned.
- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/server/integrationServerName/rest-apis/rest-apiName/
  messageflows
  ```

  If the command is successful, an HTTP status code 200 and information about the message flows that are deployed in the REST API project, if any, are returned.

Displaying message flows in an integration service project

About this task

You can display the message flows that are deployed to an integration server in an integration service project.

Procedure

Display the message flows.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/services/integrationServiceName/messageflows
  ```

  If the command is successful, an HTTP status code 200 and information about the message flows that are deployed in the integration service project, if any, are returned.
- For an integration server that is managed by an integration node:

  ```
  GET http://hostname:port/apiv2/servers/integrationServerName/services/integrationServiceName/
  messageflows
  ```

  If the command is successful, an HTTP status code 200 and information about the message flows that are deployed in the integration service project, if any, are returned.

Starting message flows by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to start message flows that are deployed to an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering message flows by using the administration REST API” on page 417

About this task

This topic describes:

- “Starting all message flows in an application” on page 421
- “Starting a specific message flow in an application” on page 422
- “Starting all message flows in a REST API project” on page 423
- “Starting a specific message flow in a REST API project” on page 423
- “Starting all message flows in an integration service project” on page 424

- "Starting a specific message flow in an integration service project" on page 424

Starting all message flows in an application

About this task

You can start all the message flows that are deployed to an integration server in an application.

Procedure

Start the message flows.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/messageflows/start
  ```

  For example, to start the message flows in application HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
  messageflows/start
  ```

  For example, to start the message flows that are in the library called HTTPStaticLib that is associated with an application called HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/libraries/
  HTTPStaticLib/messageflows/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  messageflows/start
  ```

  For example, to start the message flows in application HTTPInputApplication that is deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  libraries/libraryName/messageflows/start
  ```

  For example, to start the message flows that are in the library called HTTPStaticLib associated with an application called HTTPInputApplication that is deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  libraries/HTTPStaticLib/messageflows/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Starting a specific message flow in an application

About this task

You can start a specific message flow that is deployed to an integration server in an application.

Procedure

Start a message flow.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageFlowName/
  start
  ```

  For example, to start a message flow called `RequestServiceFlow` in an application called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows/
  RequestServiceFlow/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
  messageflows/messageFlowName/start
  ```

  For example, to start a message flow called `RequestServiceFlow` that is in a library called `HTTPStaticLib` that is associated with an application called `HTTPInputApplication`, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/libraries/
  HTTPStaticLib/messageflows/RequestServiceFlow/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  messageflows/messageFlowName/start
  ```

  For example, to start a message flow called `RequestServiceFlow` in application `HTTPInputApplication` that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  messageflows/RequestServiceFlow/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  libraries/libraryName/messageflows/messageFlowName/start
  ```

  For example, to start a message flow called `RequestServiceFlow` that is in a library called `HTTPStaticLib` associated with an application called `HTTPInputApplication` that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  libraries/HTTPStaticLib/messageflows/RequestServiceFlow/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Starting all message flows in a REST API project

About this task

You can start all the message flows that are deployed to an integration server in a REST API project.

Procedure

Start the message flows.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2//rest-apis/rest-apiName/libraries/libraryName/messageflows/
  start
  ```

  If the command is successful, an HTTP status code 200 and response message OK are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
  messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
  libraries/libraryName/messageflows/start
  ```

  If the command is successful, an HTTP status code 200 and response message OK are returned.

Starting a specific message flow in a REST API project

About this task

You can start a specific message flow that is deployed to an integration server in a REST API project.

Procedure

Start a message flow.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageFlowName/start
  ```

  If the command is successful, an HTTP status code 200 and response message OK are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
  messageflows/messageFlowName/start
  ```

  If the command is successful, an HTTP status code 200 and response message OK are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
  messageflows/messageFlowName/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
  libraries/libraryName/messageflows/messageFlowName/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Starting all message flows in an integration service project

About this task

You can start all the message flows that are deployed to an integration server in an integration service project.

Procedure

Start the message flows.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
  messageflows/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/messageflows/start
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/libraries/libraryName/messageflows/start
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Starting a specific message flow in an integration service project

About this task

You can start a specific message flow that is deployed to an integration server in an integration service project.

Procedure

Start the message flow.

- For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/messageFlowName/
start
```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
messageflows/messageFlowName/start
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.
- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/messageflows/messageFlowName/start
```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/libraries/libraryName/messageflows/messageFlowName/start
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping message flows by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to stop message flows that are deployed to an integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Administering message flows by using the administration REST API” on page 417

About this task

This topic describes:

- “Stopping all message flows in an application” on page 425
- “Stopping a specific message flow in an application” on page 426
- “Stopping all message flows in a REST API project” on page 427
- “Stopping a specific message flow in a REST API project” on page 428
- “Stopping all message flows in an integration service project” on page 429
- “Stopping a specific message flow in an integration service project” on page 429

Stopping all message flows in an application

About this task

You can stop all the message flows that are deployed to an integration server in an application.

Procedure

Stop the message flows.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/messageflows/stop
  ```

  For example, to stop the message flows in application HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows/stop
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
  messageflows/stop
  ```

  For example, to stop the message flows that are in the library called HTTPStaticLib that is associated with an application called HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/libraries/
  HTTPStaticLib/messageflows/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.
- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  messageflows/stop
  ```

  For example, to stop the message flows in application HTTPInputApplication that is deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  messageflows/stop
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  libraries/libraryName/messageflows/stop
  ```

  For example, to stop the message flows that are in the library called HTTPStaticLib associated with an application called HTTPInputApplication that is deployed on integration server ACESERV1, which is managed by an integration node, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  libraries/HTTPStaticLib/messageflows/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping a specific message flow in an application

About this task

You can stop a specific message flow that is deployed to an integration server in an application.

Procedure

Stop a message flow.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageFlowName/
  stop
  ```

  For example, to stop a message flow called RequestServiceFlow in an application called HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows/
  RequestServiceFlow/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/applications/applicationName/libraries/libraryName/
  messageflows/messageFlowName/stop
  ```

  For example, to stop a message flow called RequestServiceFlow that is in a library called HTTPStaticLib that is associated with an application called HTTPInputApplication, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/libraries/
  HTTPStaticLib/messageflows/RequestServiceFlow/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.
- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  messageflows/messageFlowName/stop
  ```

  For example, to stop a message flow called RequestServiceFlow in application HTTPInputApplication that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  messageflows/RequestServiceFlow/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
  libraries/libraryName/messageflows/messageFlowName/stop
  ```

  For example, to stop a message flow called RequestServiceFlow that is in a library called HTTPStaticLib associated with an application called HTTPInputApplication that is deployed on integration server ACESERV1, use the following curl command:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
  libraries/HTTPStaticLib/messageflows/RequestServiceFlow/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping all message flows in a REST API project

About this task

You can stop all the message flows that are deployed to an integration server in a REST API project.

Procedure

Stop the message flows.

- For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/stop
```

If the command is successful, an HTTP status code 204 is returned. No response message is returned.

If the message flows are in a library:

```
POST http://hostname:port/apiv2//rest-apis/rest-apiName/libraries/libraryName/messageflows/
stop
```

If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/stop
```

If the command is successful, an HTTP status code 204 is returned. No response message is returned.

If the message flows are in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/stop
```

If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping a specific message flow in a REST API project

About this task

You can stop a specific message flow that is deployed to an integration server in a REST API project.

Procedure

Stop a message flow.

- For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageFlowName/stop
```

If the command is successful, an HTTP status code 204 is returned. No response message is returned.

If the message flow is in a library:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/libraries/libraryName/
messageflows/messageFlowName/stop
```

If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageFlowName/stop
```

If the command is successful, an HTTP status code 204 is returned. No response message is returned.

If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
libraries/libraryName/messageflows/messageFlowName/stop
```

If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping all message flows in an integration service project

About this task

You can stop all the message flows that are deployed to an integration server in an integration service project.

Procedure

Stop the message flows.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/stop
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
  messageflows/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/messageflows/stop
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flows are in a library:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/
  services/integrationServiceName/libraries/libraryName/messageflows/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Stopping a specific message flow in an integration service project

About this task

You can stop a specific message flow that is deployed to an integration server in an integration service project.

Procedure

Stop the message flow.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/messageFlowName/
  stop
  ```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flow is in a library:

  ```
  POST http://hostname:port/apiv2/services/integrationServiceName/libraries/libraryName/
  messageflows/messageFlowName/stop
  ```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/messageflows/messageFlowName/stop
```

  If the command is successful, an HTTP status code 204 is returned. No response message is returned.

  If the message flow is in a library:

```
POST http://hostname:port/apiv2/servers/integrationServerName/
services/integrationServiceName/libraries/libraryName/messageflows/messageFlowName/stop
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Tearing down a message flow by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to tear down named message flows that are deployed to an integration server. The message flow to tear down must already be deployed on the integration server.

Before you begin

- "Managing resources by using the administration REST API" on page 334
- "Administering message flows by using the administration REST API" on page 417

About this task

The action of tearing down terminates all connections and threads that are associated with the message flow. The message flow remains in the runtime image of the integration server and can be restarted by using a start action without the need to redeploy. There is no need to stop and restart the integration server to restart the message flow.

This topic describes:

- "Tearing down a message flow in an application" on page 430
- "Tearing down a message flow in a REST API project" on page 431
- "Tearing down a message flow in an integration service project " on page 431

Tearing down a message flow in an application

About this task

You can tear down a message flow that is deployed to an integration server in an application.

Procedure

Tear down a message flow.

- For an independent integration server:

```
POST http://hostname:port/apiv2/applications/applicationName/messageflows/messageFlowName/
teardown
```

  For example, to tear down the message flow called `RequestServiceFlow` in application `HTTPInputApplication`, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/applications/HTTPInputApplication/messageflows/
RequestServiceFlow/teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/applications/applicationName/
messageflows/messageFlowName/teardown
```

  For example, to tear down the message flow called `RequestServiceFlow` in application `HTTPInputApplication` deployed to integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/servers/ACESERV1/applications/HTTPInputApplication/
messageflows/RequestServiceFlow/teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Tearing down a message flow in a REST API project

About this task

You can tear down a message flow that is deployed to an integration server in a REST API project.

Procedure

Tear down a message flow.

- For an independent integration server:

```
POST http://hostname:port/apiv2/rest-apis/rest-apiName/messageflows/messageFlowName/teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.
- For an integration server that is managed by an integration node:

```
POST http://hostname:port/apiv2/servers/integrationServerName/rest-apis/rest-apiName/
messageflows/messageFlowName/teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Tearing down a message flow in an integration service project

About this task

You can tear down a message flow that is deployed to an integration server in an integration service project.

Procedure

Tear down a message flow.

- For an independent integration server:

```
POST http://hostname:port/apiv2/services/integrationServiceName/messageflows/messageFlowName/
teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.
- For an integration server that is managed by an integration node:

```
POST /apiv2/servers/integrationServerName/services/integrationServiceName/
messageflows/messageFlowName/teardown
```

  If the command is successful, an HTTP status code 200 and response message 0K are returned.

Using resources statistics through the administration REST API

You can use the IBM App Connect Enterprise administration REST API to administer the collection of resource statistics on an integration server.

About this task

Use the administration REST API to perform the following tasks:

- Display the properties that relate to the collection of resource statistics.
- Update the properties that relate to the collection of resource statistics.
- Start collecting resource statistics.
- Stop collecting resource statistics.

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser, on GitHub, or in the documentation that is provided as part of your App Connect Enterprise installation. For more information on how to access this information, see "Managing resources by using the administration REST API" on page 334.

Displaying the properties that relate to resource statistics collection by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to display the properties that relate to resource statistics collection on an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using resources statistics through the administration REST API" on page 432

Procedure

Use one of the following REST API methods to query the resource statistics properties of an integration server.

- For an independent integration server:

  ```
  GET http://hostname:port/apiv2/statistics/resource-stats
  ```

  For example, use the following curl command:

  ```
  curl -X GET http://hostname:port/apiv2/statistics/resource-stats
  ```

  If the command is successful, an HTTP status code 200 and a response similar to the following are returned:

  ```
  {
    "hasChildren": false,
    "name": "resource-stats",
    "type": "resourceStats",
    "uri": "/apiv2/statistics/resource-stats",
    "properties": {
      "name": "Resource",
      "outputFormat": "inherit",
      "reportingOn": true,
      "type": "Resource"
    },
    "descriptiveProperties": {},
    "active": {
      "reportingOn": true
    },
    "actions": {
      "unavailable": {
  ```

```
      "start-collection": "/apiv2/statistics/resource-stats/start-collection"
    },
    "available": {
      "stop-collection": "/apiv2/statistics/resource-stats/stop-collection"
    }
  },
  "children": {},
  "links": []
}
```

- For an integration server that is managed by an integration node:

```
GET http://hostname:port/apiv2/servers/integrationServerName/statistics/resource-stats
```

For example, to display the resource statistics properties on integration server ACESERV1, which is managed by an integration node, use the following curl command:

```
curl -X GET http://hostname:port/apiv2/servers/ACESERV1/statistics/resource-stats
```

If the command is successful, an HTTP status code 200 and a response similar to the following are returned.:

```
{
  "hasChildren": false,
  "name": "resource-stats",
  "type": "resourceStats",
  "uri": "/apiv2/servers/ACESERV1/statistics/resource-stats",
  "properties": {
    "name": "Resource",
    "outputFormat": "inherit",
    "reportingOn": true,
    "type": "Resource"
  },
  "descriptiveProperties": {},
  "active": {
    "reportingOn": true
  },
  "actions": {
    "unavailable": {
      "start-collection": "/apiv2/servers/ACESERV1/statistics/resource-stats/start-
collection"
    },
    "available": {
      "stop-collection": "/apiv2/servers/ACESERV1/statistics/resource-stats/stop-collection"
    }
  },
  "children": {},
  "links": []
}
```

Updating properties that relate to resource statistics collection by using the administation REST API

You can use the IBM App Connect Enterprise administration REST API to update and persist the resource statistics properties for an integration server dynamically, without having to restart the integration server.

Before you begin

Read the following topics:

- “Managing resources by using the administration REST API” on page 334
- “Using resources statistics through the administration REST API” on page 432

About this task

You can dynamically update the properties that relate to resource statistics collection by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` subdirectory of the integration server's working directory, which means that the updates are not lost if the integration server is stopped.

Procedure

Use one of the following REST API methods to update the resource statistics properties of an integration server.

- For an independent integration server:

  ```
  PATCH http://hostname:port/apiv2/statistics/resource-stats
  ```

  For example, use the following curl command:

  ```
  curl -X PATCH http://hostname:port/apiv2/statistics/resource-stats
  ```

  The following example shows how to enable and persist a value of `false` on the **`reportingOn`** property. Setting **`reportingOn`** to `false` turns resource statistics off.

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  localhost:port/apiv2/statistics/resource-stats -d '{  "properties": { "reportingOn": "false"
    }}'
  ```

  If the command is successful, an HTTP status code 204 is returned.

  You can verify the successful setting of the property by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432, or by looking in the `server.conf.yaml` file in the `overrides` subdirectory of the integration server's working directory to see that the modified value is persisted.

- For an integration server that is managed by an integration node:

  ```
  PATCH http://hostname:port/apiv2/servers/integrationServerName/statistics/resource-stats
  ```

  For example, to enable and persist a value of `false` on the **`reportingOn`** property on integration server ACESERV1, which is managed by an integration node:

  ```
  curl -X PATCH -H 'accept:application/json' -H 'content-type:application/json' http://
  hostname:port/apiv2/servers/ACESERV1/statistics/resource-stats -d '{  "properties":
   { "reportingOn": "false"  }}'
  ```

  If the command is successful, an HTTP status code 204 is returned.

  You can verify the successful setting of the property by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432 , or by looking in the `server.conf.yaml` file in the `overrides` subdirectory to see that the modified value is persisted.

Starting resource statistics collection by using the administation REST API

You can use the IBM App Connect Enterprise administration REST API to start the collection of resource statistics for an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using resources statistics through the administration REST API" on page 432

Procedure

Use one of the following REST API methods to start resource statistics collection on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/statistics/resource-stats/start-collection
  ```

  For example, use the following curl command:

  ```
  curl -X POST  http://hostname:port/apiv2/statistics/resource-stats/start-collection
  ```

  If the command is successful, an HTTP status code 200 is returned.

  You can verify that resource statistics collection has started by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432; the value of the **reportingOn** property is `true` when resource statistics collection is started.
- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/statistics/resource-stats/
  start-collection
  ```

  For example, to start resource statistics collection on integration server ACESERV1, which is managed by an integration node:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/statistics/resource-stats/start-
  collection
  ```

  If the command is successful, an HTTP status code 200 is returned.

  You can verify that resource statistics collection has started by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432; the value of the **reportingOn** property is `true` when resource statistics collection is started.

Stopping resource statistics collection by using the administation REST API

You can use the IBM App Connect Enterprise administration REST API to stop the collection of resource statistics for an integration server.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "Using resources statistics through the administration REST API" on page 432

Procedure

Use one of the following REST API methods to stop resource statistics collection on an integration server.

- For an independent integration server:

  ```
  POST http://hostname:port/apiv2/statistics/resource-stats/stop-collection
  ```

  For example, use the following curl command:

  ```
  curl -X POST  http://hostname:port/apiv2/statistics/resource-stats/stop-collection
  ```

  If the command is successful, an HTTP status code 200 is returned.

  You can verify that resource statistics collection has stopped by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432; the value of the **reportingOn** property is `false` when resource statistics collection is stopped.

- For an integration server that is managed by an integration node:

  ```
  POST http://hostname:port/apiv2/servers/integrationServerName/statistics/resource-stats/stop-collection
  ```

  For example, to stop resource statistics collection on integration server ACESERV1, which is managed by an integration node:

  ```
  curl -X POST http://hostname:port/apiv2/servers/ACESERV1/statistics/resource-stats/stop-collection
  ```

  If the command is successful, an HTTP status code 200 is returned.

  You can verify that resource statistics collection has stopped by issuing a GET API call as described in "Displaying the properties that relate to resource statistics collection by using the administration REST API" on page 432; the value of the **reportingOn** property is `false` when resource statistics collection is stopped.

Setting message flow user-defined properties at run time by using the administration REST API

You can change the behavior of a message flow at run time by using the IBM App Connect Enterprise administration REST API to modify the values of user-defined properties on the message flow.

Before you begin

Read the following topics:

- "Managing resources by using the administration REST API" on page 334
- "User-defined properties" on page 569

About this task

When a message flow node references a user-defined property, it searches these levels of the message flow:

- If the message flow has the user-defined property named as a dynamic UDP override, this takes precedence over any other definitions.
- If the message flow node is in a subflow, and the subflow has the named user-defined property, this is used.
- If the message flow has the named user-defined property, this is used.
- If the integration server has the named user-defined property, this is used.

You can use the administration REST API to apply and remove user-defined property overrides. These overrides are not persistent, which means that they must be reapplied after the integration server is restarted.

The following REST API methods are provided for applying and removing user-defined property overrides:

- `HTTP POST to /apiv2/servers/{server}/applications/{application}/messageflows/{messageflow}/apply-global-udp-override?name=NAME&value=VALUE&type=TYPE`
- `HTTP POST to /apiv2/servers/{server}/applications/{application}/messageflows/{messageflow}/remove-global-udp-override?name=NAME`

The REST API classes and methods are described in the App Connect Enterprise REST API V2 specification, which you can view in a browser by specifying the address of your integration node or server, followed by the administration REST API port, and then `/apidocs`. For example, to display the REST API specification for an integration server, enter a URL as shown in the following example (specifying either `http` or `https`):

```
https://green.company.com:7600/apidocs
```

where `green.company.com` is the address of the integration server, and `7600` is the admin REST API port specified in the `server.conf.yaml` file.

The REST API specification is also provided with your installation of App Connect Enterprise, in the following directories:

- *install_dir*`\server\nodejs\node_modules\@ibm-app-connect\ace-admin-server\node_modules\@ibm-app-connect\ace-admin-api\docs\server`
- *install_dir*`\server\nodejs\node_modules\@ibm-app-connect\ace-admin-server\node_modules\@ibm-app-connect\ace-admin-api\docs\node`

where *install_dir* is the directory in which App Connect Enterprise is installed.

See the following topics for more information about setting user-defined property overrides:

- "Applying a user-defined property override" on page 437
- "Removing a user-defined property override" on page 438
- "Example: setting user-defined property overrides" on page 439

Applying a user-defined property override

You can apply a user-defined property override by using the IBM App Connect Enterprise administration REST API.

Before you begin

Read the following topics:

- "User-defined properties" on page 569
- "Setting message flow user-defined properties at run time by using the administration REST API" on page 436

About this task

You can use the `apply-global-udp-override` method to create an override value for a named user-defined property. When an override value is created, the value is seen anywhere that the user-defined property is referenced in the message flow, both in the main flow and in subflows, including:

- ESQL: DECLARE myVar EXTERNAL datatype defaultValue
- Java: MbNode.getUserDefinedAttribute()
- .NET: NBNode::GetUserDefinedProperty
- Mapping node access to user defined properties

User-defined property overrides are not persistent, which means that they must be reapplied after the integration node or server is restarted.

The `apply-global-udp-override` action takes the following query parameters:

Parameter	Description
name*	The name of the user-defined property for which an override is to be applied. This parameter is required.

Parameter	Description
value	The value of the user-defined property override. If the value includes characters that are not supported in REST URIs, these characters must be URI encoded. For example, a space in a value must be encoded as `%20`. This parameter is optional; if it is not supplied, the default for the datatype is used.
type	The datatype of the user-defined property override (matching ESQL datatypes). This parameter is optional; if it is omitted, the datatype of the existing override is used. If this parameter is omitted and there is no existing override, `CHARACTER` is used by default.

The following examples show how to use the `apply-global-udp-override` method:

```
apiv2/applications/UDPAPP/messageflows/FLOWWITHUDP/apply-global-udp-override?
name=myCharUDP&value=NewValue&type=CHARACTER
```

```
apiv2/applications/UDPAPP/messageflows/FLOWWITHUDP/apply-global-udp-override?
name=myBoolUDP&value=TRUE&type=BOOLEAN
```

```
apiv2/applications/UDPAPP/messageflows/FLOWWITHUDP/apply-global-udp-override?
name=myIntUDP&value=350&type=INTEGER
```

```
apiv2/applications/UDPAPP/messageflows/FLOWWITHUDP/apply-global-udp-override?
name=myCharUDP&value=NewValue
```

```
apiv2/applications/UDPAPP/messageflows/FLOWWITHUDP/apply-global-udp-override?
name=myCharUDP&value=New%20Value%20with%20spaces
```

For more information about how to set user-defined property overrides, see “Example: setting user-defined property overrides” on page 439.

You can also remove a user-defined property override by using the IBM App Connect Enterprise administration REST API, as described in “Removing a user-defined property override” on page 438.

Removing a user-defined property override

You can remove a user-defined property override by using the IBM App Connect Enterprise administration REST API.

Before you begin

Read the following topics:

- “User-defined properties” on page 569
- “Setting message flow user-defined properties at run time by using the administration REST API” on page 436

About this task

You can use the `remove-global-udp-override` method to remove an override value for a named user-defined property.

The `remove-global-udp-override` action takes the following query parameters:

Parameter	Description
name*	The name of the user-defined property for which an override is to be removed. This parameter is required. This parameter must match the name of the override that was used on the `apply-global-udp-override` method when the override was created. When an override has been removed, the message flow reverts to the value that was defined in the deployed message flow definitions, before any override was applied.

You can also use the administration REST API to apply a user-defined property override, as described in "Applying a user-defined property override" on page 437.

For more information about how to set user-defined property overrides, see "Example: setting user-defined property overrides" on page 439.

Example: setting user-defined property overrides

You can change the behavior of a message flow at run time by using the IBM App Connect Enterprise administration REST API to modify the values of user-defined properties on the message flow.

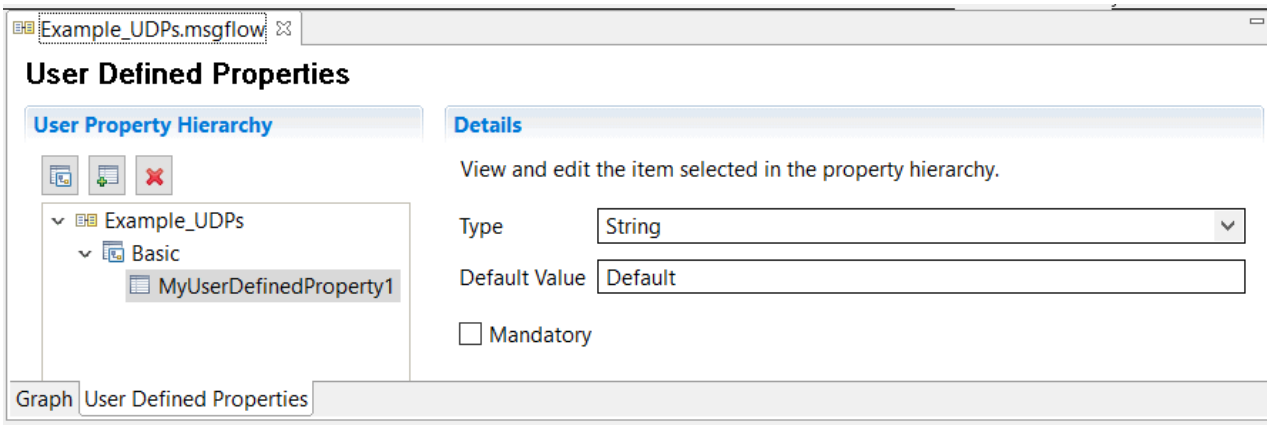
Before you begin

Read the following topics:

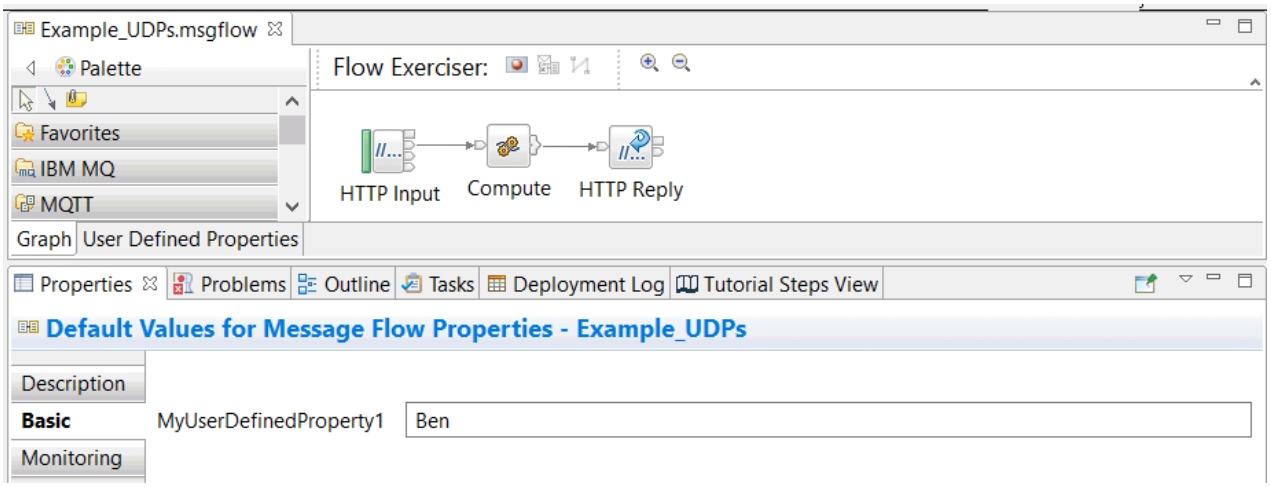
- "Setting message flow user-defined properties at run time by using the administration REST API" on page 436
- "User-defined properties" on page 569

About this task

In this example, the User Defined Properties tab in the IBM App Connect Enterprise Toolkit shows the default value of a user-defined property called `MyUserDefinedProperty1`, which is `Default`:



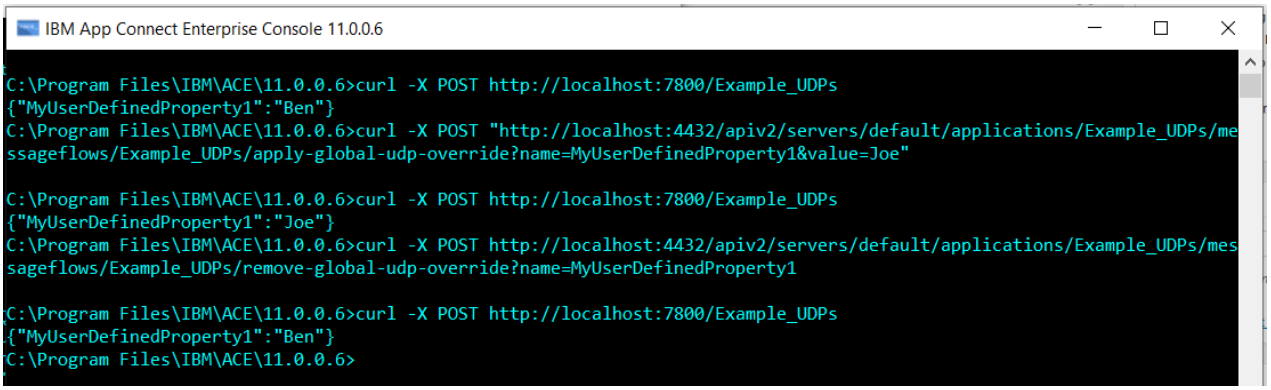
The value assigned to the property before deploying the message flow is then set to Ben:



This message flow uses a simple ESQL instruction to assign the current value of the user-defined property to an output JSON message, which is sent as an HTTP reply from the message flow:

```
DECLARE MyUserDefinedProperty1 EXTERNAL CHARACTER 'Default';

CREATE COMPUTE MODULE Example_UDPs_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        CREATE LASTCHILD OF OutputRoot DOMAIN('JSON');
        SET OutputRoot.JSON.Data.MyUserDefinedProperty1 = MyUserDefinedProperty1;
        RETURN TRUE;
    END;
```

When this message has been deployed, you can query the current value of the user-defined property and apply a new value to it, by using a simple HTTP client program such as Curl:



For information about applying and removing user-defined property overrides, see “Applying a user-defined property override” on page 437 and “Removing a user-defined property override” on page 438.

For more information about using the administration REST API, see “Managing resources by using the administration REST API” on page 334.

Updating the HTTPS Connector resource manager by using the administration REST API

You can use the IBM App Connect Enterprise administration REST API to query and update resource managers that are available to the integration server. This topic describes how to use the administration

REST API to update the HTTPS Connector integration server resource manager with particular focus on reloading the inbound security certificate.

Before you begin

Read the following topic:

- "Managing resources by using the administration REST API" on page 334

About this task

When receiving secured inbound connections over HTTPS, integration servers use the HTTPS Connector resource manager. You can query and update the HTTPS Connector resource manager by using the App Connect Enterprise administration REST API. You do not have to restart the integration server for such updates to take effect.

There are a number of situations that might require updates to the properties of the HTTPS Connector resource manager in an integration server. You can dynamically update the properties in question by using the PATCH verb in the administration REST API. Updates that are made by using the PATCH verb are persisted in the `overrides` sub-directory of the integration server working directory meaning that, if the integration server is restarted, the updates are not lost.

This topic describes:

- "Querying the available resource managers of an integration server " on page 441
- "Querying the properties of the HTTPS Connector resource manager " on page 441
- "Dynamically updating the properties of the HTTPS Connector resource manager " on page 442
- "Dynamically loading a new server certificate by reusing the same keystore file " on page 443

Querying the available resource managers of an integration server

Procedure

Use one of the following REST API methods to query the available resource managers of an integration server.

- For an independent integration server:

```
GET http://hostname:port/apiv2/resource-managers
```

- For a managed integration server:

```
GET http://hostname:port/apiv2/servers/{server}/resource-managers
```

Querying the properties of the HTTPS Connector resource manager

Procedure

Use one of the following REST API methods to query the properties of the HTTPS Connector resource manager.

- For an independent integration server:

```
GET http://hostname:port/apiv2/resource-managers/https-connector
```

  For example, use the curl command:

```
curl -X GET http://hostname:port/apiv2/resource-managers/https-connector
```

  A response similar to the following is returned:

```
{"hasChildren":false,"name":"https-connector","type":"resourceManager","uri":"/apiv2/
resource-managers/https-connector",
"properties":
{"AutoRespondToHTTPHEADRequests":false,"CORSAllowCredentials":false,"CORSAllowHeaders":"Accept,Accept-
Language,
Content-Language,Content-
Type","CORSAllowMethods":"GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS","CORSAllowOrigins":"*",
"CORSEnabled":false,"CORSExposeHeaders":"Content-Type","CORSMaxAge":-1,"CipherSpec":"!
RC4+RSA:HIGH:+MEDIUM:+LOW",
"ConnBacklog":100,"EnableLookups":true,"EnableTLSTrace":false,"IWATimeout":300,"IntegratedWindowsAuthentic
"KeyAlias":"","KeyPassword":"********","KeystoreFile":"","KeystorePassword":"********","KeystoreType":"JKS
"ListenerAddress":"0.0.0.0","ListenerPort":7843,"ListenerThreads":-1,"MaxConnections":-1,"MaxKeepAliveRequ
"MaxPostSize":-1,"QueueCapacity":1000,"ReqClientAuth":false,"ServerName":"","TLSCertVerifyDepth":100,"TLSC
"TLSProtocols":"","TimeoutSweepInterval":20,"TruststoreFile":"","TruststorePassword":"********","Truststor
"name":"HTTPSConnector","type":"Policy"},"descriptiveProperties":
{"className":"HTTPSConnector","isDynamic":"true",
"policyType":"HTTPSConnector"},"active":
{"ListenerPort":-1,"serverRestartRequired":false},"actions":{"available":{"refresh-tls-
config":
"/apiv2/resource-managers/https-connector/refresh-tls-config"}},"children":{},"links":[]}
```

- For a managed integration server:

```
GET http://hostname:port/apiv2/servers/{server}/resource-managers/https-connector
```

  For example, use the curl command:

```
curl -X GET http://hostname:port/apiv2/servers/{server}/resource-managers/https-connector
```

Dynamically updating the properties of the HTTPS Connector resource manager

Procedure

Use one of the following REST API methods to dynamically update the properties of the HTTPS Connector resource manager.

Being able to dynamically update these properties, in particular **KeystoreFile** is useful in the situation where you are using certificate authorities that refresh their certificates on a frequent basis but you do not want to restart the integration server whenever the refresh occurs.

Updates take effect on the first inbound connection following the update; preexisting connections retain the original configuration. You do not have to restart the integration server for the changes to take effect.

You can update the following HTTPS Connector resource manager properties:

- **KeyAlias**
- **KeystoreFile** - you can specify the path to an updated inbound server certificate without the need to restart the integration server for the certificate to be applied on subsequent inbound connections.
- **KeystorePassword**
- **KeystoreType**
- **TruststoreFile**
- **TruststorePassword**
- **TruststoreType**

For more information about these parameters, see Integration server HTTP listener parameters (SOAP and HTTP nodes).

- For example, to update the **KeyAlias**, **KeystoreFile**, **KeystorePassword**, and **KeystoreType** properties for the HTTPS Connector resource manager for an independent integration server, use the following curl command with the PATCH verb:

```
curl -X PATCH -H "Content-Type: application/json" http://hostname:port/apiv2/resource-
managers/https-connector
-d "{\"properties\": {\"KeyAlias\":\"server1cert\",\"KeystoreFile\": \"C:\\temp\
\myserver_keystore.p12\",\
"KeystorePassword\": \"server::keystorePass\",\"KeystoreType\": \"p12\"}}"
```

- For a managed integration server, to update the same properties for the HTTPS Connector resource manager for a managed integration server, use the following curl command with the PATCH verb:

```
curl -X PATCH -H "Content-Type: application/json" http://hostname:port/apiv2/servers/
{server}/resource-managers/https-connector
-d "{\"properties\": {\"KeyAlias\":\"server1cert\",\"KeystoreFile\": \"C:\\temp\
\myserver_keystore.p12\",\"KeystorePassword\": \
"server::keystorePass\",\"KeystoreType\": \"p12\"}}"
```

If the command is successful, an HTTP status code 204 is returned. You can see the updated values in the`overrides` sub-directory of the integration server working directory.

Dynamically loading a new server certificate by reusing the same keystore file

About this task

You can reuse the same keystore file but with a different value for the server security certificate to refresh the server certificate to be used by the integration server without having to restart the integration server. Instead of updating the server certificate by using the PATCH verb in the REST API, as described in “Dynamically updating the properties of the HTTPS Connector resource manager ” on page 442, you must use the POST method to load a new server certificate that you have defined in a certificate file with the same name as one that you have previously used.

Procedure

Use the following commands to refresh the integration server TLS configuration, including the updated server certificate value in the reused certificate file.

Updates take effect on the first inbound connection following the update. You do not have to restart the integration server for the changes to take effect.

- For example, to ensure the use of an updated server certificate specified in a reused certificate file for an independent integration server, use the following curl command:

```
curl -X POST http://hostname:port/apiv2/resource-managers/https-connector/refresh-tls-config
```

- For a managed integration server:

```
curl -X POST http://hostname:port/apiv2/servers/{server}/resource-managers/https-connector/
refresh-tls-config
```

If the command is successful, an HTTP status code 200 is returned.

Managing resources by using the IBM Integration API

Develop Java applications that use the IBM Integration API to manage integration nodes, integration servers, and their associated resources.

Before you begin

Understand how you can use the IBM Integration API to manage the resources associated with your integration nodes and servers. For more information, see #unique_95.

Samples are provided to demonstrate typical IBM Integration API scenarios. Run and explore the samples to learn about what you can do with the IBM Integration API; for more information, see "The IBM Integration API samples" on page 467.

About this task

You can use the IBM Integration API to develop administrative applications or to develop message flows.

To develop administrative applications that run tasks in your integration nodes and servers, see the following information:

- Configuring the environment
- "Connecting to an integration node from a custom integration application" on page 447
- Navigating integration node and integration node resources
- Deploying resources
- "Managing integration nodes by using JavaCompute nodes" on page 454

Results

When you have created your applications, test their operation in your test environment. Check the results of the operations that the programs have performed by using the Deployment Log view.

What to do next

When you have written and tested your applications, distribute them to the computers from which you want your administrators to perform the tasks that you have developed.

Configuring an environment for developing and running custom integration applications

Prepare the environment in which you want to run your custom integration applications.

Before you begin

To develop and run Java applications that use the IBM Integration API, you must install the following prerequisite software in your local computer environment:

- An IBM Java Development Kit (JDK) at a supported Java level. Java support is defined in the IBM App Connect Enterprise system requirements; see IBM App Connect Enterprise system requirements.

  IBM App Connect Enterprise does not supply a JDK; you must acquire and install a suitable product yourself.

  You must use an IBM JDK to develop custom integration applications and an IBM Java Runtime Environment (JRE) to run these applications.

About this task

Follow the instructions for the appropriate environment:

- "Configuring the command-line environment to run custom integration applications" on page 445

- "Configuring the Eclipse environment to run custom integration applications" on page 446

You can also run custom integration applications, and therefore control one or more integration node components, from computers on which you have not installed IBM App Connect Enterprise. For more information, see "Configuring environments without IBM App Connect Enterprise installed" on page 446.

Configuring the command-line environment to run custom integration applications

Use the command line on your computers to configure the environment to run your custom integration applications.

About this task

To run your custom integration applications with your chosen operating system, you must update the CLASSPATH environment variable:

Procedure

1. Add the IBM Integration API JAR files to your CLASSPATH.
   To add the JAR files to your CLASSPATH:
   - Linux UNIX On Linux and UNIX:

```
export CLASSPATH = $CLASSPATH%;$install_dir/server/classes/IntegrationAPI.jar:
$install_dir/common/jackson/lib/jackson-annotations-2.9.8.jar:
$install_dir/common/jackson/lib/jackson-core-2.9.8.jar:
$install_dir/common/jackson/lib/jackson-databind-2.9.8.jar:
$install_dir/common/jackson/lib/jackson-dataformat-yaml-2.9.8.jar:
$install_dir/common/jackson/lib/snakeyaml-1.23.jar:
$install_dir/common/jetty/lib/jetty-io.jar:
$install_dir/common/jetty/lib/jetty-util.jar:
$install_dir/common/jetty/lib/websocket-api.jar:
$install_dir/common/jetty/lib/websocket-client.jar:
$install_dir/common/jetty/lib/websocket-common.jar:
$install_dir/common/jetty/lib/jetty-client.jar:
$install_dir/common/jetty/lib/jetty-http.jar:
```

     - For local connections on Linux and UNIX, also add:

```
$install_dir/common/jnr/lib/jnr-a64asm-1.0.0.jar:
$install_dir/common/jnr/lib/jnr-constants-0.9.12.jar:
$install_dir/common/jnr/lib/jnr-enxio-0.21.jar:
$install_dir/common/jnr/lib/jnr-ffi-2.19.jar:
$install_dir/common/jnr/lib/jnr-posix-3.0.50.jar:
$install_dir/common/jnr/lib/jnr-unixsocket-0.23.jar:
$install_dir/common/jnr/lib/jnr-x86asm-1.0.2.jar:
$install_dir/common/jnr/lib/asm-7.0.jar:
```

   - Windows On Windows:

```
set CLASSPATH = %CLASSPATH%;%install_dir%\server\classes\IntegrationAPI.jar
%install_dir\common\jackson\lib\jackson-annotations-2.9.8.jar;
%install_dir\common\jackson\lib\jackson-core-2.9.8.jar;
%install_dir\common\jackson\lib\jackson-databind-2.9.8.jar;
%install_dir\common\jackson\lib\jackson-dataformat-yaml-2.9.8.jar;
%install_dir\common\jackson\lib\snakeyaml-1.23.jar;
%install_dir\common\jetty\lib\jetty-io.jar;
%install_dir\common\jetty\lib\jetty-util.jar;
%install_dir\common\jetty\lib\websocket-api.jar;
%install_dir\common\jetty\lib\websocket-client.jar;
%install_dir\common\jetty\lib\websocket-common.jar;
%install_dir\common\jetty\lib\jetty-client.jar;
%install_dir\common\jetty\lib\jetty-http.jar;
```

2. Add your Java development directory to the CLASSPATH in the same way.

What to do next

Use the tools that are provided by your JDK to build and run your custom integration applications.

Configuring the Eclipse environment to run custom integration applications

Use Eclipse facilities to configure the environment to run your custom integration applications.

About this task

To run your custom integration applications by using Eclipse, you must update the CLASSPATH environment variable:

Procedure

1. In your Eclipse environment, select **File** > **New** > **Project**.

   The **New Project** wizard opens.
2. Select **Java Project** from the options displayed.

   Click **Next**. The **New Java Project** window opens.
3. Enter a name for your new project.

   Click **Next**.
4. Select the **Libraries** tab, and click **Add External Jars**.
5. To set up the build environment, navigate to the *install_dir*/common/classes subdirectory. For example, for Version 12.0 on Windows operating systems, navigate to the directory `C:\Program Files\IBM\ACE\12.0.n.0\common\classes`.

   Select the file `IntegrationAPI.jar`, and click **Open**. The file is added to the list in the window for the **Libraries** tab.
6. When you have added the file `IntegrationAPI.jar` to the build path, set up the runtime environment.

   Click **Add External Jars** again, and navigate to the *install_dir*/common/jackson/lib subdirectory.

   Select each of the JAR files in that directory, and click **Open**. These files are also added to the list.
7. For local connections, also add *install_dir*/common/jnr/lib.
8. Click **Finish**.

What to do next

Use the Eclipse development tools to build and run your custom integration applications.

Configuring environments without IBM App Connect Enterprise installed

Install and run custom integration applications and other utilities on computers, when you are using a local ID only, on which you have not installed IBM App Connect Enterprise.

About this task

You can run Java applications that use the IBM Integration API even where you have not installed IBM App Connect Enterprise. These applications include your own custom integration applications, and some of the command utilities.

To install Java applications that use the IBM Integration API in an environment that does not have IBM App Connect Enterprise installed, complete the following steps:

Procedure

1. Ensure that the target computer has a compatible Java Runtime Environment (JRE).

   Because you are not installing IBM App Connect Enterprise, which includes a JRE, you must use an alternative option.

   Note, that you must also set MQSI_JREPATH to the installation path of your JRE.

   Java support is defined in the IBM App Connect Enterprise system requirements; see IBM App Connect Enterprise system requirements.
2. Copy the following set of files from a computer that has IBM App Connect Enterprise installed to the target computer:
   a. The `IntegrationAPI.jar` file from the `install_dir/common/classes` directory.
   b. All the `.jar` files from the `install_dir/common/jackson/lib` directory.
   c. All the `.jar` files from the `install_dir/common/jnr/lib` directory.
   d. Your custom integration applications and all configuration files, for example `.broker` files.
3. On the target computer, use system facilities to update the CLASSPATH environment variable to include the following files:
   - The JAR file that contains the definitions of the IBM Integration API classes: `IntegrationAPI.jar`.
   - The JAR files that you copied from the `install_dir/common/jackson/lib` directory and the `install_dir/common/jnr/lib` directory.
   - Your applications that import the IBM Integration API classes.
   - The JAR files that contain the definitions of the `IntegrationNodeConnectionParameters` classes.
   - Any other required JAR files and directories. For example, if you require any of the available command utilities on the target computer, include `brokerutil.jar`; if you require the integration node (BIP) messages to be displayed in locales other than US English, include a directory that contains BIPmsgs*.properties.
4. Ensure that the user ID that the target computer uses has the following authorities:
   - Authority to manipulate integration node objects.

What to do next

You can run your custom integration applications, and the supported command utilities, on the target computer.

Connecting to an integration node from a custom integration application

Connect an application that uses the IBM Integration API to an integration node, to send requests about its status and its resources.

Before you begin

You must complete the steps in “Configuring an environment for developing and running custom integration applications” on page 444.

About this task

Review the following example application, `IntegrationNodeStateChecker.java`, which demonstrates how to use the IBM Integration API classes:

```
import com.ibm.integration.admin.proxy.*;

public class IntegrationNodeStateChecker {
```

```
 public static void main(String[] args) {
  // The IP address of where the integration node is running
  // and the web administration port number of the integration node.
  displayNodeRunState("localhost", 4414);
 }

 public static void displayNodeRunState(String hostname, int port) {

  IntegrationNodeProxy node = new IntegrationNodeProxy(hostname,port,"","",false);

  try {
   String nodeName = node.getName();

   System.out.println("Integration node '"+nodeName+
     "' is available!");
  } catch (IntegrationAdminException ex) {
   System.out.println("Integration node is NOT available"+
     " because "+ex);
  }
 }
}
```

This example application connects to a remote integration node.

Review each part of the application that uses the IBM Integration API to understand what is required to connect to an integration node. Use the following steps to assist you in creating your own custom integration applications.

Procedure

1. Import the `com.ibm.integration.admin.proxy` package, which contains the IBM Integration API Java classes.

   This action is in the first line of the application:

   ```
   import com.ibm.integration.admin.proxy.*;
   ```

The first line of code inside the `try` block of the `displayNodeRunState()` method instantiates an `IntegrationNodeProxy` object. `IntegrationNodeProxy` is an interface that states that implementing classes are able to provide the parameters to connect to an integration node.

The `IntegrationNodeConnectionParameters` class implements this interface by defining a set of HTTP connection parameters. The constructor that is used in the program has two required parameters:

a. The host name of the computer that the integration node is running on.

b. The web administration port (the default is 4414). For more information about port numbers for an integration node, see “Configuring the IBM App Connect Enterprise web user interface” on page 87.

If you have administration security enabled, use constructors for the user name and password. For example:

```
IntegrationNodeConnectionParameters node =
 new IntegrationNodeConnectionParameters(hostname,port,"userid","password",false);
```

2. The `getName()` method, and other methods that request information from the integration node, cause a block until the information is supplied, or a timeout occurs. Therefore, if the integration node is not running, the application hangs for a period. You can control the timeout period by using the `BrokerProxy.setRetryCharacteristics()` method. Typically, blocking occurs only when a resource is accessed for the first time within an application.

Results

You can create a custom integration application that can connect to an integration node.

Navigating integration node and integration node resources in a custom integration application

Explore the status and attributes of the integration node that your custom integration application is connected to, and discover information about its resources.

Before you begin

You must have completed "Connecting to an integration node from a custom integration application" on page 447.

About this task

Each resource that the integration node can control is represented as a single object in the IBM Integration API. A custom integration application can request status and other information about the following objects:

- Integration nodes
- Integration servers
- Deployed message flows

The IBM Integration API also handles deployed message sets; these resources are handled as attributes of deployed integration servers.

Collectively known as *administered objects*, these objects provide most of the interface to the integration node, and are therefore fundamental to an understanding of the IBM Integration API.

Each administered object is an instance of a Java class that describes the underlying type of object in the integration node. The main Java classes are shown in the following table.

Java class	Class function
ApplicationProxy	Describes applications that have been deployed to integration servers.
IntegrationNodeProxy	Describes integration nodes.
IntegrationServerProxy	Describes integration servers.
MessageFlowProxy	Describes message flows that have already been deployed to integration servers through an application, REST API, or integration service; it does not describe message flows in the Integration Development perspective of the IBM App Connect Enterprise Toolkit.
RestApiProxy	Describes the REST API application details that have been deployed to an integration server.
ServiceProxy	Describes the integration service details that have been deployed to an integration server.
SharedLibraryProxy	Describes the shared library details that have been deployed to an integration server.
StaticLibraryProxy	Describes the static library details that have been deployed to an integration server through an application, REST API, or integration service.
SubFlowProxy	Describes subflows that have already been deployed to integration servers through an application, REST API, or integration service.

Each administered object describes a single object that can be controlled by the integration node. For example, every integration server within an integration node has one `IntegrationServerProxy` instance that represents it within the application.

A set of public methods is available for each administered object, which applications can use to inquire and manipulate properties of the underlying integration node to which the instance refers. To access an administered object through its API, your application must first request a handle to that object from the object that logically owns it.

For example, because integration nodes logically own integration servers, to gain a handle to integration server EG1 running on integration node B1, the application must ask the `IntegrationNodeProxy` object represented by B1 for a handle to the `IntegrationServerProxy` object represented by EG1.

On a `IntegrationNodeProxy` object that refers to integration node B1, the application can call methods that cause the integration node to reveal its run-state, or cause it to start all its message flows. You can write applications to track the changes that are made to the integration node objects by reading the messages maintained as `LogEntry` objects.

In the following example, a handle is requested to the `IntegrationNodeProxy` object. The `IntegrationNodeProxy` is logically the root of the administered object tree, therefore your application can access all other objects in the integration node directly, or indirectly.

The integration node directly owns the integration servers, therefore applications can call a method on the `IntegrationNodeProxy` object to gain a handle to the `IntegrationServerProxy` objects. Similarly, the integration server logically contains the set of all message flows, therefore the application can call methods on the `IntegrationServerProxy` object to access the `MessageFlowProxy` objects.

Each object with an asterisk after its name inherits from the DeployedObjectGroupProxy class, and therefore has further child objects.

The following application traverses the administered object hierarchy to discover the run-state of a deployed message flow. The application assumes that message flow MF1 is deployed to EG1 on integration node B1; you can substitute these values in the code for other values that are valid in the integration node.

```
import com.ibm.integration.admin.proxy.*;

public class GetMessageFlowRunState {

  public static void main(String[] args) {

    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);

    System.out.println("Getting message flow status of MF1 in the default application");
      displayMessageFlowRunState(node, "Server1", "MF1");
    }

    private static void displayMessageFlowRunState(
                                IntegrationNodeProxy node,
                                String serverName,
                                String flowName) {
    try {
      IntegrationServerProxy server =
        node.getIntegrationServerByName(serverName);

      if (server != null) {
        MessageFlowProxy mf =
          server.getDefaultApplication(true).getMessageFlowByName(flowName,"",false);
          /* locate the message flow from the default application. boolean flag indicates */
          /* if we need to fresh the model or use the cache one from proxy. */

        if (mf != null) {
          boolean isRunning = mf.getMessageFlowModel(false).getActive().isRunning();
          System.out.print("Flow "+flowName+" on " +
            serverName+" on "+server.getName()+" is ");

          if (isRunning) {
            System.out.println("running");
          } else {
            System.out.println("stopped");
          }
        } else {
          System.err.println("No such flow "+flowName);
        }
      } else {
        System.err.println("No such integration server "+serverName+"!");
      }
```

```
    } catch(IntegrationAdminException ex) {
      System.err.println("Comms problem! "+ex);
    }
  }
}
```

The method `displayMessageFlowRunState()` does most of the work. This method takes the valid `IntegrationNodeProxy` handle gained previously, and discovers the run-state of the message flow in the following way:

1. The `IntegrationNodeProxy` instance is used to gain a handle to the `IntegrationServerProxy` object with the name described by the string `serverName`
2. If a valid integration server is returned, the `IntegrationServerProxy` instance is used to gain a handle to the `MessageFlowProxy` object with the name described by the string `flowName`.
3. If a valid message flow is returned, the run-state of the `MessageFlowProxy` object is queried, and the result is displayed.

The application does not have to know the names of objects that it can manipulate. Each administered object contains methods to return sets of objects that it logically owns. The following example demonstrates this technique by looking up the names of all integration servers within the integration node.

```
import java.util.List;

import com.ibm.integration.admin.proxy.IntegrationAdminException;
import com.ibm.integration.admin.proxy.IntegrationNodeProxy;
import com.ibm.integration.admin.proxy.IntegrationServerProxy;

public class DisplayIntegrationServerNames {

  public static void main(String[] args) {

    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);

    System.out.println("Get the Integration Server Names");
    displayIntegrationServerNames(node);
   }
private static void displayIntegrationServerNames(IntegrationNodeProxy node)
  {
    try {
      List<IntegrationServerProxy> serverNames = node.getAllIntegrationServers();
      if (serverNames != null)
      {
        for(IntegrationServerProxy name: serverNames)
          {
          System.out.println("Found : "+name.getName());
          }
        }
      else
      { // no servers found or a Rest call returned a bad response
        // check the status of the last http response
        int status = node.getLastHttpResponse().getStatusCode();
      if (node.getLastHttpResponse().getStatusCode() == 200)
        {
          System.out.println("No servers Found");
        }
      else
        {
          System.out.println("Http response from Integration Node is "+status);
          System.out.println("Http response reason: "+node.getLastHttpResponse().getReason());
        }
      }

      } catch(IntegrationAdminException ex) {
      System.err.println("Comms problem! "+ex);
    }
  }
}
```

The properties of any proxy are obtained by calling a `getxxxxModel` method call, where `xxx` is the current proxy object being used. The model class is a representation of the JSON response message

returned from the REST calls. The following example shows how to obtain the process identifier of an integration node and an integration server:

```
import com.ibm.integration.admin.model.IntegrationNodeModel;
import com.ibm.integration.admin.model.IntegrationServerModel;
import com.ibm.integration.admin.proxy.*;


public class GetProperties {

  public static void main(String[] args) {

    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);
    try {
      IntegrationNodeModel nodeModel = node.getIntegrationNodeModel(true);
      System.out.println("Node Process ID: "+nodeModel.getActive().getProcessId());

      IntegrationServerProxy server = node.getIntegrationServerByName("Server1");
      IntegrationServerModel serverModel = server.getIntegrationServerModel(true);
      System.out.println("Server Process ID: "+serverModel.getActive().getProcessId());
    } catch (IntegrationAdminException e) {
      System.out.println("Error connecting: "+e);
    }
  }
}
```

Each proxy class caches the last model that was obtained from the REST call. Subsequent calls return the cached value from the proxy. If you want the model to be updated, set the refresh flag on the call to `true`, which causes the proxy to discard the cached version and obtain the latest properties.

Deploying resources to an integration node by using a custom integration application

Deploy BAR files to the integration nodes in your integration node network by using a custom integration application.

About this task

You can also check the result of a deployment by using the IBM Integration API.

To use the IBM Integration API to deploy files and check the results:

Procedure

1. Connect to the integration node by clicking **File** > **Connect to Local Integration node** or **File** > **Connect to Remote Integration node**.

   This action opens the Connect to Integration node dialog.
2. Enter the relevant connection parameters in the dialog.

   A hierarchical representation of the integration node and its resources is displayed.
3. Complete one or more of the following operations:
   - Click an object in the tree to display the attributes of that object.
   - Right-click an object in the tree to call IBM Integration API methods that manipulate that object. For example, right-clicking an integration node opens a menu that includes the items **Start user trace**.
   - Use the log pane to view useful information that relates to the operation in progress.

Example

The following example custom integration application connects to an integration node that is running on the local computer. The integration node is listening for RestAdminListener requests on the web administration port (the default is 4414). The code deploys a BAR file that is called `MyBAR.bar` to an integration server called `default` that is running on the integration node, and displays the result.

```
import com.ibm.integration.admin.model.common.LogEntry;
```

```
import com.ibm.integration.admin.model.server.DeployResult;
import com.ibm.integration.admin.proxy.IntegrationAdminException;
import com.ibm.integration.admin.proxy.IntegrationNodeProxy;
import com.ibm.integration.admin.proxy.IntegrationServerProxy;


public class DeployExample {

 public static void main(String[] args) {
  IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);
  try {
   IntegrationServerProxy server = node.getIntegrationServerByName("server1");
   DeployResult result  = server.deploy("C:\\temp\\mybarfile.bar");
   int deployStatus = server.getLastHttpResponse().getStatusCode();
   if (deployStatus >= 400)
   {
    System.out.println("Http response error: " + deployStatus );
   }
   else
   { // deploy worked, check for any Log entries
    LogEntry[] logEntries  = result.getLogEntry();
    if (logEntries != null)
    {
     for (LogEntry logEntry: logEntries)
     {
      System.out.println("Deploy log entry: "+logEntry.getDetailedText());
     }
    }
   }
  } catch (IntegrationAdminException e) {
   System.out.println("Error connecting: "+e);
  }
 }
}
```

Checking the results of deployment in a custom integration application

When you deploy from a custom integration application, you can check the results of that action.

About this task

Code your application to test the results of the deployment actions that it takes. For example:

```
import com.ibm.integration.admin.model.common.LogEntry;
import com.ibm.integration.admin.model.server.DeployResult;
import com.ibm.integration.admin.proxy.IntegrationAdminException;
import com.ibm.integration.admin.proxy.IntegrationNodeProxy;
import com.ibm.integration.admin.proxy.IntegrationServerProxy;


public class DeployExample {

  public static void main(String[] args) {
    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);
    try {
      IntegrationServerProxy server = node.getIntegrationServerByName("server1");
      DeployResult result  = server.deploy("C:\\temp\\mybarfile.bar");
      int deployStatus = server.getLastHttpResponse().getStatusCode();
      if (deployStatus >= 400)
      {
        System.out.println("Http response error: " + deployStatus );
      }
      else
      { // deploy worked, check for any Log entries
        LogEntry[] logEntries  = result.getLogEntry();
        if (logEntries != null)
        {
          for (LogEntry logEntry: logEntries)
          {
            System.out.println("Deploy log entry: "+logEntry.getDetailedText());
          }
        }
      }
    } catch (IntegrationAdminException e) {
      System.out.println("Error deploying: "+e);
    }
  }
}
```

If the deployment message could not be sent to the integration node, an `IntegrationAdminException` exception is thrown at the time of the deployment. If the integration node receives the deployment message, log messages for the overall deployment are displayed, followed by completion codes specific to each integration node that is affected by the deployment.

Managing integration nodes by using JavaCompute nodes

You can use the IBM Integration API to manage integration nodes and their associated resources from JavaCompute nodes in deployed message flows.

Before you begin

You must create a node in a message flow.

About this task

Use IBM Integration API methods and classes in your JavaCompute node to explore and manage integration nodes and other resources.

Procedure

1. Create the Java class for the node in which you want to include IBM Integration API methods.
2. Add the IBM Integration API JAR file (`IntegrationAPI.jar`), which is in the *`install_dir`*`/common/classes/` directory, to the Java build path for the associated Java project.
3. To work with the integration node, add the following constructor to your code:

   ```
   IntegrationNodeProxy node = new IntegrationNodeProxy();
   ```

   This method returns an instance of the IntegrationNodeProxy object for the integration node to which the message flow (that contains this node) is deployed.
4. To work with an integration server on this integration node, or an independent integration server, add the following constructor to your code:

   ```
   IntegrationServerProxy server = new IntegrationServerProxy();
   ```

   This method returns an instance of the IntegrationServerProxy object for the integration server to which the message flow is deployed.
5. If you want to connect to a different integration node on the computer to which your node and message flow are deployed, you can use a variant of this class:

   ```
   IntegrationNodeProxy node = new IntegrationNodeProxy(string);
   ```

   Specify the name of the alternative local integration node as the value of the variable *string*.
   Your code can manage this second integration node, and its associated resources, by using the IntegrationNodeProxy object that is returned by this call.
6. Include additional IBM Integration API methods in your Java code to run the operations that you want against the integration node or integration server by using the objects that are obtained by following the previous steps.

   You can follow the guidance that is provided in other topics in this section for further information and examples that show how to use IBM Integration API methods in custom integration applications.

   If you include methods that affect the message flow in which your custom integration application is running, it might not be able to receive all notifications that these operations are successfully complete. Stopping, deleting, and redeploying the message flow are examples in this category; consider carefully the consequences of using these methods.
7. Deploy the JAR file, and all associated message flows, in a BAR file.

   You do not have to deploy the `IntegrationAPI.jar` file to the target integration server, because the integration node can access these classes independently.

Creating objects by using a custom integration application

Create new objects associated with an integration node.

About this task

The following example adds an integration server called EG2 to the connected integration node.

```
import com.ibm.integration.admin.proxy.*;

public class AddIntegrationServer {

  public static void main(String[] args) {

    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);
    try {
      IntegrationServerProxy server = node.createIntegrationServer("EG2");
    } catch (IntegrationAdminException e) {
      System.out.println("Error connecting: "+e);
    }
  }
}
```

If the request to create the object described by the skeleton fails, all requests that use the skeleton also fail. Therefore, if integration server EG2 cannot be created, all subsequent requests that concern the skeleton object fail. However, unless the application explicitly checks for errors, it works in the same way as it does in the successful case, because no exception is thrown unless, because of a communication problem, a message cannot be sent to the integration node.

Developing applications using the IBM Integration API: Example code

Examples of IBM Integration API code for common tasks, to help you write your own Java code to develop message flow applications or modify user-defined patterns.

About this task

The IBM Integration API uses the `com.ibm.broker.appdev.*` classes in the `IntegrationAPI.jar` file.

The following topics show examples of Java code that you can use to complete common tasks:

Procedure

- “Running PHP code” on page 466
- “Saving a message flow file from memory” on page 466

Loading an existing message flow into memory

Use the IBM Integration API when developing message flow applications to load a message flow into memory. You can then access the message flow in your Java code.

You must load a message flow into memory to use it with the IBM Integration API methods within your Java code. To load a message flow into memory, create a `File` object and use the `read()` method of the `FlowRendererMSGFLOW`, which makes the message flow available for your Java code. The `read()` method takes the message flow project that contains the required message flow file and the relative path to the message flow file from this project.

The following example shows how to load a message flow that is in the `mqsi` directory:

```
import java.io.File;
import java.io.IOException;
import com.ibm.broker.MessageBrokerAPIException;
import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.FlowRendererMSGFLOW;

public class LoadMessageFlow {
    public static void main(String[] args) {
        File msgFlow = new File("../mqsi/main.msgflow");
        try {
            MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
        } catch (IOException e) {
            // Add your own code here
            e.printStackTrace();
        } catch (MessageBrokerAPIException e) {
            // Add your own code here
            e.printStackTrace();
        }
    }
}
```

Pattern authoring

When you create patterns with the pattern authoring tool, use the `getMessageFlow()` method of the `PatternInstanceManager` object, which is automatically passed to your Java code. Modify a pattern instance to load a message flow into memory and make it available to other IBM Integration API methods.

The following example shows how to load a message flow that is in the default schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject",
 "main.msgflow");
        if (mf1 != null) {
           // Message flow was found
        }
        else {
           // Message flow was not found
        }
    }
}
```

The following example shows how to load a message flow that is in the `mqsi` schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "mqsi/
main.msgflow");
        if (mf1 != null) {
           // Message flow was found
        }
        else {
           // Message flow was not found
        }
    }
```

```
}
```

Renaming a node

Use the IBM Integration API when developing message flow applications to rename a node.

To rename a node, you must first access the required node. You can access the node by using the existing name of the node and the `getNodeByName()` method. You can then rename the node by using the `setNodeName()` method, which takes the new node name as a parameter.

The following example shows how to rename a node that is named `My Input Node`:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Node mqinNode = mf1.getNodeByName("My Input Node");
mqinNode.setNodeName("New Input Node");
```

Pattern authoring

The following example shows how to rename the previous example node for a pattern:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mqinNode.setNodeName("New Input Node");
```

Adding a node and a subflow node

Use the IBM Integration API when developing message flow applications to add a new node or subflow node to a message flow.

You can add a new built-in node, or a new subflow node to a message flow.

- When you add a subflow node by using the IBM Integration API, you must link the subflow node with the subflow message flow by using the `setSubFlow()` method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

  ```
  sfNode.setSubFlow(sub1);
  ```

- If you want to set a node property on a subflow node, use the `addNodeProperty()` method to set the property before you link the node to the subflow by using the `setSubFlow()` method. If you use `addNodeProperty()` after `setSubFlow()`, the node property might not be stored.

The following example shows you how to add a new built-in node:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
mf1.addNode(mqinNode);
```

The following example shows you how to add a new subflow node to a message flow:

1. The new subflow node is added to the message flow held in object `mf1`.
2. The subflow node is linked to the subflow message flow by using the setSubFlow() method.
3. The subflow node name is set to `My Sub Flow Node`.
4. A new subflow node is created and assigned to object `sfNode`.

The subflow can be stored in a `.msgflow` format:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File subFlow = new File("subflow.msgflow");
```

```
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

The subflow can be stored in a `.subflow` format:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
sFile subFlow = new File("subflow.subflow");
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

Pattern authoring

The following example shows you how to add a new built-in node to a pattern instance:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
mf1.addNode(mqinNode);
```

The following example shows you how to add a new subflow node to a message flow:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

Setting the position of a node

Use the IBM Integration API to set the position of a node on the canvas in the Application Development view.

Set the position of a node on the canvas by using the `setLocation()` method of the node object. The following example sets the position of a new MQOutput node to coordinates x=300 pixels, y=100 pixels:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQOutputNode mqoutNode = new MQOutputNode();
mqoutNode.setLocation(300, 100);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQOutputNode mqoutNode = new MQOutputNode();
mqoutNode.setLocation(300, 100);
```

Copying a node

Use the IBM Integration API to copy a node or subflow node to a message flow.

You can copy a built-in or subflow node by using the `clone()` method. In the following example, a new MQInput node *mqinNode* is created and properties on the node are set. A new MQInput node *mqinNode1* is then created by copying *mqinNode* by using the `clone()` method. When the node is copied, the node properties are also copied.

Example:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
MQInputNode mqinNode1 = (MQInputNode) mqinNode.clone();
mqinNode1.setNodeName("Copy of My Input Node");
mf1.addNode(mqinNode1);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
MQInputNode mqinNode1 = (MQInputNode) mqinNode.clone();
mqinNode1.setNodeName("Copy of My Input Node");
mf1.addNode(mqinNode1);
```

Removing a node

Use the IBM Integration API when developing message flow applications to remove a node from a message flow.

To remove a node, you must first get the required node from the message flow object. In the following example, the `getNodeByName()` method is used to get the required node from message flow object *mf1*. The node is then removed by using the `removeNode()` method. When a node is removed, any connections to or from the node are also removed.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Node mqinNode = mf1.getNodeByName("My Input Node");
mf1.removeNode(mqinNode);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mf1.removeNode(mqinNode);
```

Adding connections between nodes

Use the IBM Integration API when developing message flow applications to add connections between nodes.

Pattern authoring

You can connect both subflow and built-in nodes. The first example shows you how to connect two built-in nodes. The second example shows you how to connect a built-in node to a subflow node.

To use the terminals of a subflow node, you must link the subflow node with the subflow message flow by using the `setSubFlow()` method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

```
sfNode.setSubFlow(sub1);
```

The following example shows you how to connect two built-in nodes:

1. A MQInput node and a Collector node are created.
2. The `getInputTerminal()` method is used to create a dynamic Input terminal called *NEWIN* on the Collector node.
3. The Input terminal is connected to the Output terminal of the MQInput node by using the `connect()` method of the message flow object *mf1*.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
CollectorNode colNode = new CollectorNode();
colNode.getInputTerminal("NEWIN");
mf1.connect(mqinNode.OUTPUT_TERMINAL_OUT, colNode.getInputTerminal("NEWIN"));
```

To use a static terminal on a node, you must use the appropriate constant defined for it in the IBM Integration API. These constants are listed in the IBM Integration API javadoc, see IBM Integration API.

The following example shows you how to connect a subflow node to a built-in node. You must load the subflow message flow and link it to a subflow node. You must use the `getInputTerminal()`, `getInputTerminals()`, `getOutputTerminal()` or `getOutputTerminals()` method to access the terminal on the subflow node to which you want to connect. The example code completes the following steps:

1. The main message flow, `main.msgflow`, and a Compute node in the main message flow are loaded into memory.
2. The subflow message flow, subflow.msgflow, and the subflow node in the main message flow are loaded into memory.
3. The `setSubFlow()` method of the subflow node is used to link the subflow message flow *sub1* to the subflow node *sfNode*.
4. The `getOutputTerminal()` method is used to get the Process terminal of the subflow node. The `connect()` method of the message flow object is used to connect this terminal to the Input terminal of the Compute node.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
ComputeNode compNode = (ComputeNode)mf1.getNodeByName("My Compute Node");
File subFlow = new File("subflow.msgflow");
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = (SubFlowNode)mf1.getNodeByName("My Subflow Node");
sfNode.setSubFlow(sub1);
mf1.connect(sfNode.getOutputTerminal("Process"), compNode.INPUT_TERMINAL_IN);
```

The following example is the same as the previous example for connecting two build-in nodes, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
CollectorNode colNode = new CollectorNode();
colNode.getInputTerminal("NEWIN");
mf1.connect(mqinNode.OUTPUT_TERMINAL_OUT, colNode.getInputTerminal("NEWIN"));
```

The following example is the same as the previous example for using static terminals on a node, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ComputeNode compNode = (ComputeNode)mf1.getNodeByName("My Compute Node");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = (SubFlowNode)mf1.getNodeByName("My Subflow Node");
sfNode.setSubFlow(sub1);
mf1.connect(sfNode.getOutputTerminal("Process"), compNode.INPUT_TERMINAL_IN);
```

Adding and connecting user-defined nodes

Use the IBM Integration API to add user-defined nodes and to connect user-defined nodes to other nodes.

The code required to add and connect user-defined node is different to the code required for built-in nodes and subflow nodes.

When you write Java code for user-defined nodes you must be aware of the following information:

- User-defined nodes are supported by instances of the `GenericNode` class. To add user-defined nodes to message flows, create instances of `GenericNode` and add them to the message flow instance.
- To retrieve existing instances of a user-defined node, call `getNodeByName()` and cast the returned object to a `GenericNode` object.
- The terminals defined on your user-defined nodes are not automatically available in the API. If you create an instance of a `GenericNode` class, it does not have any input or output terminals listed. The methods `getInputTerminals()` and `getOutputTerminals()` return empty lists.
- To get an input terminal for a `GenericNode`, call `getInputTerminal()` and pass the terminal name that exists on the generic node. This method returns the input terminal and makes it available in the message flow object that contains your generic node. After you have used `getInputTerminal()` with a known terminal name, this input terminal is returned if `getInputTerminals()` is used.
- To get an output terminal for a `GenericNode`, call `getOutputTerminal()` and pass the terminal name that exists on the generic node. This method returns the output terminal and makes it available in the message flow object that contains your generic node. After you have used `getOutputTerminal()` with a known terminal name, this output terminal is returned if `getOutputTerminals()` is used.

The following example shows how you can add a user-defined node to a message flow and connect it to a built-in node:

1. An MQInput node is created and added to the message flow.
2. A user-defined node is created by using the `GenericNode` class and is added to the message flow object.
3. The static output terminal of the MQInput is assigned to the variable *outputTerminal*.
4. The input terminal of the user-defined node is assigned to the variable *inputTerminal* by using the `getInputTerminal()` method with the known terminal name *In*.
5. The nodes are connected by using the `connect()` method.
6. The final section of code shows that the input node is now available for use in the message flow, by using the `getInputTerminals()` method of the user-defined node instance.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);

MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("IN");
mf1.addNode(mqinNode);

GenericNode myNode = new GenericNode("MyUserDefinedNode");
myNode.setNodeName("MyNode");
mf1.addNode(myNode);

OutputTerminal outputTerminal = mqinNode.OUTPUT_TERMINAL_OUT;
InputTerminal inputTerminal = myNode.getInputTerminal("In");
mf1.connect(outputTerminal, inputTerminal);

InputTerminal[] inputTerminals = myNode.getInputTerminals();
System.out.println("Input terminals on my node:");
for (int i = 0; i < inputTerminals.length; i++) {
   InputTerminal inputTerminal = inputTerminals[i];
   System.out.println(inputTerminal.getName());
}
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");

MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("IN");
mf1.addNode(mqinNode);

GenericNode myNode = new GenericNode("MyUserDefinedNode");
myNode.setNodeName("MyNode");
mf1.addNode(myNode);

OutputTerminal outputTerminal = mqinNode.OUTPUT_TERMINAL_OUT;
InputTerminal inputTerminal = myNode.getInputTerminal("In");
mf1.connect(outputTerminal, inputTerminal);

InputTerminal[] inputTerminals = myNode.getInputTerminals();
System.out.println("Input terminals on my node:");
for (int i = 0; i < inputTerminals.length; i++) {
   InputTerminal inputTerminal = inputTerminals[i];
   System.out.println(inputTerminal.getName());
}
```

Removing connections between nodes

Use the IBM Integration API to remove connections between nodes.

You can disconnect two nodes by using the `disconnect()` method of the message flow object. You must provide this method with the names of the terminal instances that you want to disconnect.

The following example shows how you can disconnect two nodes:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = (MQInputNode)mf1.getNodeByName("My Input Node");
MQOutputNode mqoutNode = (MQOutputNode)mf1.getNodeByName("My Output Node");
mf1.disconnect(mqinNode.OUTPUT_TERMINAL_OUT, mqoutNode.INPUT_TERMINAL_IN);
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = (MQInputNode)mf1.getNodeByName("My Input Node");
MQOutputNode mqoutNode = (MQOutputNode)mf1.getNodeByName("My Output Node");
mf1.disconnect(mqinNode.OUTPUT_TERMINAL_OUT, mqoutNode.INPUT_TERMINAL_IN);
```

Creating or changing user-defined properties

Use the IBM Integration API to create or change user-defined properties (UDPs).

You can create new UDPs and add them to the message flow, or you can discover existing UDPs and modify them.

The following example shows you how to create a UDP and add it to a message flow:

1. A UDP called `Property1` is created in parameter group `Group1`. The data type of the UDP is defined as a string, and the UDP is given the default value `Hello World!`
2. The UDP is then added to the message flow by using the `addFlowProperty()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
UserDefinedProperty udp = new UserDefinedProperty("Group1", "Property1",
 UserDefinedProperty.Usage.MANDATORY, UserDefinedProperty.Type.STRING, "Hello World!");
mf1.addFlowProperty(udp);
```

In the following example, the existing UDPs in a message flow are discovered by using the `getFlowProperties()` method on the message flow. The `setName()` method is then used to set the name of the first UDP to `Property3`:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Vector<FlowProperty> flowProperties = mf1.getFlowProperties();
flowProperties.get(0).setName("Property3");
```

You can also modify user-defined properties by using the administration REST API, as described in "Setting message flow user-defined properties at run time by using the administration REST API" on page 436.

Pattern authoring

The following examples are the same as the previous UDP examples, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
UserDefinedProperty udp = new UserDefinedProperty("Group1", "Property1",
 UserDefinedProperty.Usage.MANDATORY, UserDefinedProperty.Type.STRING, "Hello World!");
mf1.addFlowProperty(udp);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Vector<FlowProperty> flowProperties = mf1.getFlowProperties();
flowProperties.get(0).setName("Property3");
```

Changing pattern parameter values

Use the IBM Integration API to modify a pattern instance to change pattern parameter values.

You can change a pattern parameter value by using the `PatternInstanceManager` object.

The following example sets the value of the pattern parameter with the pattern parameter ID `pp1` to `true`:

```
patternInstanceManager.setParameterValue("pp1", "true");
```

Reading table values

Use the IBM Integration API to read table values by accessing and extracting table values.

To read table values, you must first access the table and then extract the data from each row. To read the table values, use one of the following examples:

Use this example to access and extract the table values by providing the name of the columns. In this example, *noRows* is the number of rows in the table, *value* is the parameter that stores the table value, *column* is the name of the column in the table, and *pp3* is the parameter ID.

```
PatternParameterTable paramtable = pim.getParameterTable("pp3");
int noRows = paramtable.getRowCount();
String value;
PatternParameterRow row;
for(int i=0; i<noRows; i++) {
    row = paramtable.getRow(i);
    value = row.getValue("column");
    //insert your code here
}
```

Alternatively, if you do not provide the name of the columns, you can use this example to access and extract the table values by using the `getColumns()` method. In this example, *noColumns* is the number of columns in the table, *value* is the parameter that stores the table value, *columns* is an array containing the names of the columns in the table, and *pp3* is the parameter ID.

```
PatternParameterTable paramtable = pim.getParameterTable("pp3");
int noRows = paramtable.getRowCount();
PatternParameterRow row;
```

```
String[] columns;
String value;
for(int i=0; i<noRows; i++) {
    row = paramtable.getRow(i);
    columns = row.getColumns();
    int noColumns = columns.length;
    for(int j=0; j<noColumns; j++) {
        value = row.getValue(columns[j]);
        //insert your code here
    }
```

Creating ESQL modules

Use the IBM Integration API when developing message flow applications to create ESQL modules. You can then associate ESQL modules with nodes that use ESQL. For example, if you create a Compute node, you can write Java code to create an ESQL module and then associate that module with the node.

Examples of nodes that use ESQL are Compute, Database, DatabaseInput, and Filter nodes. If you are using a non-default broker schema, you must set the schema by using the `setBrokerSchema()` method.

The following example shows you how to create an ESQL module in the `mqsi` schema. The module is then assigned to a new Compute node by using the `setComputeExpression()` method:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
ESQLModule module = new ESQLModule();
module.setBrokerSchema("mqsi");
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

The following example shows you how to create an ESQL module in the default schema. The `setBrokerSchema()` method is not required.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

The following example shows you how to discover an ESQL module from within an ESQL file by using the `getEsqlModules()` method. You can then use the ESQL module to set the compute expression on a Compute node by using the `setComputeExpression()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

Pattern authoring

The following examples are the same as the three previous examples, but tailored for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setBrokerSchema("mqsi");
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
```

```
mf1.addNode(compNode);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

Renaming a message flow

Use the IBM Integration API when developing message flow applications to rename message flows.

You can write code to rename a message flow by using the `setName()` method. In the following example, a message flow file called `main.msgflow` is renamed to `mainGenerated.msgflow`.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
mf1.setName(mf1.getName()+"Generated");
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
mf1.setName(mf1.getName()+"Generated");
```

Updating filter tables on Route nodes

Use the IBM Integration API to update filter tables on Route nodes.

You can add and update rows on the filter table of a Route node.

Adding a new row

The following example shows you how to add a new row to a filter table by using the `createRow()` method:

1. The message flow and the Route node are loaded into memory.
2. The filter table of the Route node is loaded into memory by using the `getFilterTable()` method of the RouteNode object.
3. A new filter table row is created by using the `createRow()` method.
4. The value of the `filter pattern` property on this new row is set to `value="123"` by using the `setFilterPattern()` method.
5. The `routing output terminal` property is set to NEWOUT by using the `setRoutingOutputTerminal()` method.
6. The new row is then added to the filter table by using the `addRow()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
RouteNode.FilterTableRow newRow = filterTable.createRow();
newRow.setFilterPattern("value=\"123\"");
newRow.setRoutingOutputTerminal("NEWOUT");
filterTable.addRow(newRow);
```

Updating a row

The following example shows you how to update rows on the filter table of a Route node.

1. The message flow, Route node, and filter table of the Route node are loaded into memory.
2. The rows of the filter table are loaded into memory by using the `getRows()` method.
3. The `filter pattern` property of the first row of the filter table is set to `value2="456"`.
4. The `routing output terminal` property of the first row of the filter table is set to `NEWOUT2`.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
Vector<RouteNode.FilterTableRow> filterTableRows = filterTable.getRows();
filterTableRows.get(0).setFilterPattern("value2=\"456\"");
filterTableRows.get(0).setRoutingOutputTerminal("NEWOUT2");
```

Pattern authoring

The following examples are the same as the previously described examples, but for pattern authoring:

Adding a new row

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
RouteNode.FilterTableRow newRow = filterTable.createRow();
newRow.setFilterPattern("value=\"123\"");
newRow.setRoutingOutputTerminal("NEWOUT");
filterTable.addRow(newRow);
```

Updating a row

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
Vector<RouteNode.FilterTableRow> filterTableRows = filterTable.getRows();
filterTableRows.get(0).setFilterPattern("value2=\"456\"");
filterTableRows.get(0).setRoutingOutputTerminal("NEWOUT2");
```

Running PHP code

Use the IBM Integration API to modify a pattern instance to run PHP scripts from within your Java code.

You can run PHP scripts from within the IBM Integration API by using the `runScript()` method of the PatternInstanceManager object. In the following example, the IBM Integration API runs a PHP script in the `com.your.company.domain.code` plug-in, `templates/script.php`.

```
@Override
public void onGeneratePatternInstance (PatternInstanceManager patternInstanceManager) {
 patternInstanceManager.runScript("com.your.company.domain.code", "templates/script.php");
}
```

Saving a message flow file from memory

Use the IBM Integration API when developing message flow applications to save a message flow from memory.

Save a message flow to a message flow file to preserve additions and modifications that you have made. To save a message flow from memory, use the `write()` method of the `FlowRendererMSGFLOW` object. The `write()` method writes all the changes made using the IBM Integration API to the specified message flow file.

The schema for a message flow is stored in the message flow, and cannot be modified by using the IBM Integration API. When you save your message flow file by using the IBM Integration API, the file is saved to a directory that has the same name as the schema. If the directory does not exist, it is created. For

example, if your message flow is part of the `mqsi` schema, when you save the message flow file it is saved to the `mqsi` directory.

The following example shows how to save a message flow in the specified directory by using the `write()` method. On Windows, if the message flow is part of the `mqsi` schema, the message flow file is saved to `C:\MB\mqsi\main.msgflow`.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
FlowRendererMSGFLOW.write(mf1,"C:\MB");
```

The IBM Integration API samples

Explore the samples to learn the basic features that are provided by the IBM Integration API.

Deploy BAR

The Deploy BAR sample deploys a BAR file to an integration server, and displays the outcome. Read about this sample in "Running the IBM Integration API Deploy BAR file sample" on page 467.

Modify IBM Integration API samples

You can modify the IBM Integration API samples, and change various parameters that affect how the samples run. Read "Modifying the IBM Integration API samples" on page 468 and follow the guidance given about possible changes.

Running the IBM Integration API Deploy BAR file sample

Run the IBM Integration API Deploy BAR file sample to deploy a BAR file to an integration server.

Before you begin

- You must have an integration node and an integration server that are started.
- You must have a BAR file.

About this task

Use the *Deploy BAR file* sample to deploy a BAR file to an integration server, and display the outcome. The *Deploy BARfile* sample consists of one file:

- Windows On Windows: *install_dir*`\server\sample\admin\IntegrationAPI\DeployBAR.java`
- Linux UNIX On other platforms: *install_dir*`/server/sample/admin/IntegrationAPI/DeployBAR.java`.

Procedure

1. Run the *Deploy BAR file* sample by compiling and then running `DeployBAR.java` for your platform, with the parameters *host_name*, *port_number*, *int_server_name*, and *bar_file_name*,

   where:

   host_name
   Specifies the host name of the integration node; for example `localhost`.

   port_number
   Specifies the port number of the integration node; for example 4414.

   int_server_name
   Specifies the name of the integration server; for example `default`.

   bar_file_name
   Specifies the name of the BAR file; for example `mybar.bar`.

   The IBM Integration API connects to the integration node by using the host name and port that are specified. Next, the IBM Integration API deploys the specified BAR file to the specified integration server. The following message is displayed to confirm the actions.

   ```
   Connecting to the integration node running at host_name:port_number...
   Discovering integration server 'int_server_name'...
   Deploying bar_file_name...
   Result = success
   ```

2. Check the results of the sample by viewing the integration node in the IBM App Connect Enterprise Toolkit.

Modifying the IBM Integration API samples

Modify the IBM Integration API samples to change the parameters that they use to complete their tasks.

About this task

If you change a sample, you must recompile the source code to implement those changes. For example, you might choose to change the default connection parameters. If you modify these and other values, you change the behavior of the sample.

To modify a sample, perform the following steps:

Procedure

1. Set up the environment by following the instructions provided in "Configuring an environment for developing and running custom integration applications" on page 444.
2. Locate the source file for the sample that you want to change.

   The source files for the samples are located in the following directories:

   Deploy BAR sample
   install_dir`/server/sample/admin/IntegrationAPI/DeployBar.java`

   Integration node info sample
   install_dir`/server/sample/admin/IntegrationAPI/IntegrationNodeInfo.java`

   Integration node operations sample
   install_dir`/server/sample/admin/IntegrationAPI/IntegrationNodeOperations.java`
3. Open the source file, and modify the appropriate parameters.
4. Save and recompile the source file.

What to do next

Run the modified sample by following the instructions in the appropriate link:

- “Running the IBM Integration API Deploy BAR file sample” on page 467

Managing resources by using the IBM App Connect Enterprise web user interface

Use the IBM App Connect Enterprise web user interface to manage resources.

Before you begin

Understand how you can access the IBM App Connect Enterprise web user interface. For more information, see “Accessing the web user interface” on page 89.

About this task

You can use the IBM App Connect Enterprise web user interface to manage resources. For more information, see the following topics:

- “Applying a user-defined property override in the IBM App Connect Enterprise web user interface” on page 469.
- “Removing a user-defined property override in the IBM App Connect Enterprise web user interface” on page 470.
- “Update the message flow thread pool by using the IBM App Connect Enterprise web user interface” on page 472.

Applying a user-defined property override in the IBM App Connect Enterprise web user interface

You can apply a user-defined property override by using the IBM App Connect Enterprise web user interface.

Before you begin

Read the following topics:

- “User-defined properties” on page 569
- “Setting message flow user-defined properties at run time by using the administration REST API” on page 436

About this task

User-defined property overrides are not persistent, which means that they must be reapplied after the integration node or integration server is restarted.

You can also remove a user-defined property override by using the IBM App Connect Enterprise administration REST API, as described in “Removing a user-defined property override in the IBM App Connect Enterprise web user interface” on page 470.

Procedure

1. Start the web user interface, by using one of the following methods:
   - **From the IBM App Connect Enterprise Toolkit:** In the toolkit, right-click the integration node or server and select **Start web user interface** from the menu.
   - **From a web browser:** Enter the URL for the web user interface into a web browser, in the following form:

     protocol`://`*hostname*`:`*port*

     where:

     – *protocol* is either `http` or `https`
     – *hostname* is the host that you specified in the **`host`** property in the `.yaml` configuration file for the integration node or server (for example, `localhost` or `127.0.0.1`)
     – *port* identifies the port that is to be used for the web user interface and the toolkit, and must match the value that you set for the **`port`** property in the `.yaml` configuration file. By default the port is set to 4414.

     Examples:

     ```
     https://localhost:4414
     ```

     or

     ```
     https://127.0.0.1:4414
     ```
2. In the web user interface, click the **Open List of Options** icon for the required integration server, and select **Open** from the drop-down menu.
   - If you are viewing all the available servers under the **Servers** tab, the **Open List of Options** icon appears within the tile for each integration server.
   - If you are viewing the content of an integration server under the **Contents** tab, the **Open List of Options** icon appears in the title bar for the integration server.
3. Click the **Open List of Options** icon for the required application, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each deployed application.
4. Click the **Open List of Options** icon for the required message flow, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each message flow.
5. Select the **Properties** tab for the message flow. The properties tab has a list of properties and a list of user-defined properties.
6. Click **Override user-defined-property** to open the **Override user-defined-property** wizard.
7. In the drop-down menu in the **Override user-defined-property** wizard, select the user-defined property that you want to override.
8. In the **Value** field, type in the new value for the user-defined property.
9. Click **Apply**.

   The new value appears in the list of user-defined properties. The **Remove override** button appears next to the **Override user-defined-property** button. The **Override user-defined-property** wizard closes.

Removing a user-defined property override in the IBM App Connect Enterprise web user interface

You can remove a user-defined property override by using the IBM App Connect Enterprise web user interface.

Before you begin

Read the following topics:

- "User-defined properties" on page 569
- "Setting message flow user-defined properties at run time by using the administration REST API" on page 436

About this task

You can also apply a user-defined property override by using the IBM App Connect Enterpriseweb user interface. For more information, see "Applying a user-defined property override in the IBM App Connect Enterprise web user interface" on page 469.

Procedure

1. Start the web user interface, by using one of the following methods:
   - **From the IBM App Connect Enterprise Toolkit:** In the toolkit, right-click the integration node or server and select **Start web user interface** from the menu.
   - **From a web browser:** Enter the URL for the web user interface into a web browser, in the following form:

     protocol`://`*hostname*`:`*port*

     where:

     – *protocol* is either `http` or `https`
     – *hostname* is the host that you specified in the **`host`** property in the `.yaml` configuration file for the integration node or server (for example, `localhost` or `127.0.0.1`)
     – *port* identifies the port that is to be used for the web user interface and the toolkit, and must match the value that you set for the **`port`** property in the `.yaml` configuration file. By default the port is set to 4414.

     Examples:

     ```
     https://localhost:4414
     ```

     or

     ```
     https://127.0.0.1:4414
     ```
2. In the web user interface, click the **Open List of Options** icon for the required integration server, and select **Open** from the drop-down menu.
   - If you are viewing all the available servers under the **Servers** tab, the **Open List of Options** icon appears within the tile for each integration server.
   - If you are viewing the content of an integration server under the **Contents** tab, the **Open List of Options** icon appears in the title bar for the integration server.
3. Click the **Open List of Options** icon for the required application, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each deployed application.
4. Click the **Open List of Options** icon for the required message flow, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each message flow.
5. Select the **Properties** tab for the message flow. The properties tab has a list of properties and a list of user-defined properties.
6. Click **Remove override** to open the **Remove user-defined property override** wizard.
7. In the drop-down menu in the **Remove user-defined property override** wizard, select the user-defined property override that you want to remove.
8. Click **Remove**.

   The original value of the user-defined property appears in the list of user-defined properties. The **Remove override** button continues to appear next to the **Override user-defined-property** button

unless all of the user-defined property overrides are removed. The **Remove user-defined property override** wizard closes.

Update the message flow thread pool by using the IBM App Connect Enterprise web user interface

You can change the number of threads for a message flow by using the IBM App Connect Enterprise web user interface to update the size of the thread pool for the message flow.

Before you begin

Read the following topic:

- "Workload management" on page 2873

Procedure

1. Start the web user interface, by using one of the following methods:
   - **From the IBM App Connect Enterprise Toolkit:** In the toolkit, right-click the integration node or server and select **Start web user interface** from the menu.
   - **From a web browser:** Enter the URL for the web user interface into a web browser, in the following form:

     protocol`://`*hostname*`:`*port*

     where:

     – *protocol* is either `http` or `https`
     – *hostname* is the host that you specified in the **`host`** property in the `.yaml` configuration file for the integration node or server (for example, `localhost` or `127.0.0.1`)
     – *port* identifies the port that is to be used for the web user interface and the toolkit, and must match the value that you set for the **`port`** property in the `.yaml` configuration file. By default the port is set to 4414.

     Examples:

     ```
     https://localhost:4414
     ```

     or

     ```
     https://127.0.0.1:4414
     ```
2. In the web user interface, click the **Open List of Options** icon for the required integration server, and select **Open** from the drop-down menu.
   - If you are viewing all the available servers under the **Servers** tab, the **Open List of Options** icon appears within the tile for each integration server.
   - If you are viewing the content of an integration server under the **Contents** tab, the **Open List of Options** icon appears in the title bar for the integration server.
3. Click the **Open List of Options** icon for the required application, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each deployed application.
4. Click the **Open List of Options** icon for the required message flow, and select **Open** from the drop-down menu. The **Open List of Options** icon appears within the tile for each message flow.
5. Select the **Properties** tab for the message flow.
6. Select the **Open List of Options** icon that appears in the title bar for the message flow.
7. Click **Update thread pool** to open the **Update thread pool** wizard.
8. In the **Thread pool size** field in the **Update thread pool** wizard, type in the new value, or use the up and down arrow keys to select the new value.

9. Optional: Select the checkbox for **Start all threads immediately on flow start**.
10. Optional: Select the checkbox for **Delete and recreate the thread pool**.
11. Click **Apply**.

    The new values appear for the following properties in the **Properties** tab for the message flow:

    - *Thread capacity*
    - *Thread instances start immediately*
    - *Threads*
    - *Threads in use*

    The **Update thread pool** wizard closes.

Administering Java applications

Manage the Java applications that are deployed to an integration node.

About this task

You can deploy message flows that contain Java applications to an integration node. The Java code is run within a JVM that is created by the integration server. The JVM runs in the same process as other integration node components such as message parsing and nodes that are not Java-based.

Administration of Java applications includes the following tasks:

- "Tuning JVM parameters" on page 473
- "Configuring class loaders for Java user-defined nodes" on page 473
- "Configuring class loaders for JavaCompute nodes" on page 473
- "Configuring class loaders for ESQL routines" on page 474

Tuning JVM parameters

About this task

Use the **mqsichangeproperties** command to tune the JVM parameters to ensure that there are sufficient resources for all the Java applications that are deployed to the integration server. For more details, see command and JVM parameter values.

Configuring class loaders for Java user-defined nodes

About this task

Java user-defined nodes are manually installed onto an integration node as either a PAR file or a JAR file. Because PAR and JAR files are only loaded by the integration node on startup, the integration node must be restarted. For more details, see "Packaging a Java user-defined node " on page 2450.

A PAR file is given its own Java class loader, ensuring that the node classes are isolated from any other node classes.

For more details, see "User-defined node class loading" on page 2451.

Configuring class loaders for JavaCompute nodes

About this task

A JavaCompute node is deployed to an integration server as part of a BAR file. A JavaCompute node can specify a Java Class Loader policy to be used by the node. A Java Class Loader policy defines the behavior of the class loaders that are used by the node (see "JavaCompute node class loading" on page 1754).

If a JavaCompute node specifies a Java Class Loader policy, you must define a policy with the name that is specified by the node (see "JavaCompute node class loading by using a policy" on page 1754).

Configuring class loaders for ESQL routines

About this task

You can identify Java methods to invoke from ESQL routines by using the CREATE FUNCTION statement or the CREATE PROCEDURE statement with a LANGUAGE clause of JAVA. You use the EXTERNAL NAME clause of the statement to specify the fully-qualified class and name of the method. To find the Java class that contains a method, the integration node uses the search algorithm that is described in Deploying Java classes. You can optionally specify a Java Class Loader policy when you identify a Java method in this way. The Java Class Loader policy defines the behavior of the class loader that is used to load the class specified in the EXTERNAL NAME clause. If you include a CLASSLOADER clause in a CREATE FUNCTION statement or a CREATE PROCEDURE statement, you must ensure that the corresponding policy is defined on the integration node. For more details, see Java Class Loader policy (JavaClassLoader).

ESQL Java routines and JavaCompute nodes that specify the same Java Class Loader policy share one instance of the class loader. Therefore, the nodes and the routines use the same in-memory version of Java classes, and have access to the same static variables. The class-loading mechanism for ESQL routines is the same as for JavaCompute nodes; for more details, see "JavaCompute node class loading by using a policy" on page 1754.

Changing the location of the IBM App Connect Enterprise working directory

The IBM App Connect Enterprise working directory is the location where components store internal data, such as installation logs, component details, and trace output. The shared-classes directory is also located in the work path directory and is used for deployed Java code. If the working directory does not have enough capacity, redirect the directory to another file system that has enough capacity.

About this task

The IBM App Connect Enterprise working directory[1] is fixed at installation time so that IBM App Connect Enterprise can always find the information that it needs, and always knows where to store new information.

If you need to change the location (for example, if you do not have enough capacity on the automatically-designated file system), do not change the path to the directory; instead, redirect the original working directory to a new location.

Note: When you create an integration node, you can specify a separate working directory for that integration node. For more information about this, see the -w *workPath* parameter of the command.

Procedure

- "Changing the location of the working directory on Windows systems" on page 474
- "Changing the location of the working directory on Linux " on page 475

Changing the location of the working directory on Windows systems

About this task

When you change the location of the IBM App Connect Enterprise working directory, you mount a new partition at the location of the original working directory.

[1] This working directory is sometimes referred to as the IBM App Connect Enterprise work path directory.

To change the location of the working directory on Windows, complete the following steps:

Procedure

1. Shut down all IBM App Connect Enterprise services and processes.
2. Create a new partition on the system.
   The new partition can be on the same drive as the original working directory, or on a different drive.
3. In the IBM App Connect Enterprise Console, locate the working directory for your installation on the local system by running the following command:

   ```
   echo %MQSI_WORKPATH%
   ```
4. Copy the contents of the working directory to the new partition.
5. Delete the contents of the original working directory.
6. Open the Computer Management dialog:
   Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Computer Management**.
   The Computer Management dialog opens.
7. In the left pane of the Computer Management dialog, click **Disk Management**.
   The new partition that you added, and any existing partitions, are listed in the right pane.
8. Right-click the new partition, then click **Change Drive Letter and Paths**.
   The Change Drive Letter and Paths dialog opens.
9. Click **Add**.
   The Add Drive Letter or Path dialog opens.
10. Ensure that **Mount in the following empty NTFS folder** is selected, then browse to the original working directory location.
11. Click **OK**, then click **OK** again.

Results

Any files that IBM App Connect Enterprise creates in the working directory location are stored on the new partition.

Changing the location of the working directory on Linux

About this task

When you change the location of the IBM App Connect Enterprise working directory, you can either mount a new partition at the location of the original working directory, or you can replace the original working directory with a soft link that points to a new working directory.

To change the location of the working directory on Linux, complete the following steps:

Procedure

1. Shut down all IBM App Connect Enterprise services and processes.
2. Create a new directory on a suitable file system.
3. Locate the working directory for your installation on the local system by running the following command:

   ```
   echo $MQSI_WORKPATH
   ```
4. Copy the contents of the working directory to the new partition.
5. Delete the contents of the original working directory.

6. Perform one of the following tasks so that the IBM App Connect Enterprise installation uses the new working directory location:
   - Use the **mount** command to mount the new working directory at the location of the original working directory.
   - Delete the original working directory and replace it with a soft link. Give the soft link the same name as the original working directory and point the link to the new working directory.

Results

Any files that IBM App Connect Enterprise creates in the working directory location are stored in the new location.

Backing up the IBM App Connect Enterprise Toolkit workspace

The IBM App Connect Enterprise Toolkit workspaces contain your personal settings and data, such as message flow and message set resources. You can have multiple workspaces in different locations, and you can also have references to projects that are in other locations, therefore consider all these locations when you back up your resources.

About this task

The default workspace directory for the IBM App Connect Enterprise Toolkit depends on the platform on which it is running:

- 

On Windows, the default workspace directory is created at `C:\Users\`*`user_name`*`\IBM\ACET12\workspace`.
- 

On Linux, the default workspace directory is created at `$HOME/IBM/ACET12/workspace`.

where *user_ID* is the user name with which you are logged on. Back up files in these locations, and in all other locations in which you have saved workspace files.

The IBM App Connect Enterprise Toolkit workspace directory contains a directory called `.metadata`, which contains your personal settings and preferences for the IBM App Connect Enterprise Toolkit. If the `.metadata` directory gets corrupted, you lose these settings, and the IBM App Connect Enterprise Toolkit reverts to the default layout and preferences. If you have not backed up the `.metadata` directory, you must manually set all preferences again, and import all projects, such as message flow projects, that were displayed in the Application Development view. To back up the `.metadata` directory, take a copy of the directory.

The IBM App Connect Enterprise Toolkit workspace also contains a directory for each project (for example, a message flow project) that you have created in the IBM App Connect Enterprise Toolkit. These directories contain your data, which you must back up.

Use one of the following methods to back up the data in your workspace.

Procedure

- Export your working projects.

  You can export the projects directly as a compressed file. For further information, see *Exporting* in the Eclipse Workbench User Guide.
- Copy the project directories from the workspace directory to another location.
- Copy all your BAR files to back up their contents.

  Include all the associated source files when you build your BAR files; you can then save all content by saving only the BAR file. To add resources to a BAR file that is ready for deployment, select **Include source files**, which adds the message flow and message set source files, and the compiled files.

What to do next

If you want to restore the resources, copy the directories back into your workspace directory and import the projects. For instructions, see *Importing* in the Eclipse Workbench User Guide.

Backing up resources

Back up your integration node components, and the working files associated with integration nodes and the IBM App Connect Enterprise Toolkit, so that you can restore these resources if required.

About this task

Back up your integration node components regularly to ensure that you can return to a known operational state if necessary. In addition to configuration and operation state, the integration node maintains additional resources in its work path, and you can request that these resources are also backed up.

Back up the workspaces you have created in the IBM App Connect Enterprise Toolkit; these resources contain your application development resources; for example message flows and message model schema files. If you use a development repository to store application resources, such as Rational ClearCase, see the documentation associated with that repository to check how you can back up this data.

The following topics tell you how to back up and restore integration nodes and the IBM App Connect Enterprise Toolkit workspace:

- “Backing up the integration node” on page 477
- “Restoring the integration node” on page 478
- “Backing up the IBM App Connect Enterprise Toolkit workspace” on page 476

On distributed systems, you can use the backed up components to restore the integration node only on an identical operating environment. The operating system must be at the same level, and the integration node and queue manager names must be identical.

Backing up the integration node

Back up the integration node configuration and all associated resources.

Before you begin

Create the integration node.

About this task

You can back up the integration node and its resources to preserve the current state of the integration node configuration. You can use the backup file that is created to restore an integration node in an identical operating environment: the operating system must be at the same level, and the integration node and queue manager names must be identical.

You can run this command for an integration node that is active. However, you must not take a backup while the integration node is processing configuration changes and deployments; the backup file created might contain incomplete information. If the file contains partial records, you cannot use it to restore the integration node at a later time.

To ensure that the backup is complete and correct, take a backup either when the integration node is not processing a configuration change (such as a deployment or change property) or when the integration node is stopped.

Procedure

1. If you want to back up an active integration node, check that no configuration change requests are in progress.

   For example, if you are changing integration node properties, or have initiated a deployment, wait for these actions to complete before you back up the integration node. Active message flows are unaffected by the backup process.

   If you prefer, you can stop the integration node before you take a backup by using the **mqsistop** command.
2. Back up the integration node.

   Specify the integration node name and the location in which the backup file is created. You can also optionally specify the name of the backup file, and the name of a file to which a detailed trace is written.

   - Linux Windows UNIX Run the **mqsibackupbroker** command, specifying the integration node name and the directory to which the backup file is written.

     For example, to back up an integration node on Windows, enter the following command:

     ```
     mqsibackupbroker INODE -d c:\MQSI\BACKUP
     ```
3. When the command has completed successfully, you can continue to use the integration node.

   If you stopped the integration node, restart it by using the **mqsistart** command.

Results

The current integration node configuration is saved in the backup file. Keep the file safe so that you can restore the integration node at a later date if required.

Restoring the integration node

Restore an integration node configuration that you backed up previously.

Before you begin

Back up the integration node.

About this task

You can restore an integration node on a computer that has an identical configuration by using the backup file that you created. The operating system must be at the same level, and the integration node and queue manager names must be identical.

Procedure

1. If you have deleted the integration node and it no longer exists, or if you are restoring it on a different computer, create it by using the **mqsicreatebroker** command.

   Use the same name and parameters that you used for the integration node that you backed up.
2. If the integration node is running, stop it by using the **mqsistop** command.

   If you intend to restore common configuration information from other integration nodes that you have configured on this computer, you must also stop all the integration nodes that share this common information. For example, you can restore profile information from common files.

3. Restore the integration node.

   Specify the integration node name and the name and location of the backup file. If you want to restore common configuration information, or if you want a trace of the actions that are taken, specify the appropriate parameters for your platform.

   - 
     Run the **mqsirestorebroker** command.

     For example, to restore an integration node on Windows, enter the following command:

     ```
     mqsirestorebroker WBRK_BROKER -d c:\MQSI\BACKUP -a mybroker.zip
     ```

4. When the command has completed successfully, start the integration node by using the **mqsistart** command. If the integration node has an associated queue manager, it must be running before you restart the integration node.

What to do next

The integration node configuration has been restored; you can continue your work with this integration node.

Chapter 6. Developing integration solutions

IBM App Connect Enterprise provides a flexible environment in which you can develop integration solutions to transform, enrich, route, and process your business messages and data. You can integrate client applications that use different protocols and message formats.

About this task

In IBM App Connect Enterprise, you develop an integration solution by using an application or an integration service.

- An *application* is a container for all the resources that are required to create an integration solution.
- An *integration service* is a specialized application with a defined interface that acts as a container for a web services solution.

You can use patterns to create integration solutions. *Patterns* provide reusable solutions that encapsulate a tested approach to solving a common architecture, design, or deployment task in a particular context.

An *integration solution* is the container for the resources that you develop to process your business messages and data. IBM App Connect Enterprise manages three sets of resources to integrate your applications, messages, and data:

- Message flows
- Message flow nodes
- Message models

A message flow is a sequence of processing steps that run in an integration node when an input message is received. An integration node is a set of execution processes that host one or more message flows to route, transform, and enrich in-flight messages.

You can configure message flows to use one or both of the supported communication models, point-to-point and publish/subscribe.

You can share the project for your integration solution with others by exporting and importing the project as a compressed file known as a *project interchange file*. For example, you can export a project interchange file from the IBM App Connect Enterprise Toolkit and others can import the file into their IBM App Connect Enterprise Toolkit to share the development of an integration solution or to otherwise use that solution as the starting point for a new integration solution.

You can also copy project directories from your workspace directory to another location; for example, to backup the project directory or to use in another workspace directory.

Procedure

Read the following sections to learn more about developing integration solutions. You can also use the Tutorials Gallery in the IBM App Connect Enterprise Toolkit to explore some pre-built integration solutions.

- Learn about the development resources that you require to build your integration solution. For more information, see “Resource management overview” on page 1942.
- Learn how to develop a message flow. A message flow is a sequence of processing steps that run in an integration node when an input message is received. For more information, see “Developing message flows” on page 482.
  - Learn how to connect your business process applications and data to your message flows. You can use a number of different protocols to communicate with the integration node. You can also interact

from your message flows with other products and services. For more information, see “Connecting client applications” on page 737.

  - Learn how to design your message flows to handle messages and data in different ways. You can choose from a range of message flow nodes that support:
    - “Routing messages” on page 1233
    - “Transforming and enriching messages” on page 1249
    - “Processing events” on page 1819
    - “Handling errors in message flows” on page 1883
  - Learn how to create a message flow by running Java code that uses the IBM Integration API. For more information, see “Managing resources by using the IBM Integration API” on page 444.
- Learn about different ways of creating your integration solutions.
  - “Developing integration solutions from scratch” on page 1941
  - “Developing integration solutions by using integration services” on page 1984
  - “Developing integration solutions by using patterns” on page 2056
  - “Developing integration solutions by using REST APIs” on page 2011

  If you have a project interchange file for an existing integration solution, you can import the project for that solution into your IBM App Connect Enterprise Toolkit. In your IBM App Connect Enterprise Toolkit:

  1. Click **File** > **Import...** > **IBM Integration / Project Interchange**
  2. In the Import wizard, select the zip file (project interchange file) and the project location root (workspace directory)
  3. Click **Finish**.
- Learn how to develop unit tests for your message flows and message flow nodes. For more information, see “Developing integration tests” on page 1891.
- Learn how to define your own message models. For more information, see “Constructing message models” on page 2133.

What to do next

When you create a project, it is recommended to avoid using spaces in the project name. Although using spaces in the name is valid, you might encounter problems. For example: A scenario where an XML schema file located in project X references an XML schema file project Y, either through import or include statements. If the referenced schema in the project contain spaces in project name, the name does not resolve, and you receive errors.

Developing message flows

Develop message flows to process your business messages and data. A message flow is a sequence of processing steps that run in an integration server when an input message is received.

About this task

A message flow is a sequence of processing steps that run in the integration server when an input message is received. To learn how to design, create, and configure a message flow and its associated resources, read the following sections:

- “Message flows overview” on page 483
- “Managing message flow resources” on page 572
- “Designing a message flow” on page 583
- “Defining message flow content” on page 614

When a message is received and processed by a message flow, the integration server handles both successful and failure processing.

Procedure

IBM App Connect Enterprise provides several ways in which you can develop the message flows that you require to support your business processes requirements.

Read the following sections to learn more about developing message flows:

- "Transforming and enriching messages" on page 1249: To exchange messages between multiple applications, you might find that the applications do not understand or expect messages in the same format. You must provide some processing between the sending and receiving applications that ensures that both can continue to work unchanged, but can exchange messages successfully.
- "Routing messages" on page 1233: To route messages through your message flow, you use nodes for decision making. Alternatively, you route your messages to applications by using the publish/subscribe method of messaging.
- "Connecting client applications" on page 737: To connect your client applications, you use one or more of the supported protocols from other resources and software servers in your network.
- "Managing resources by using the IBM Integration API" on page 444: To develop Java™ message flows, you use the IBM Integration API.
- "Processing events" on page 1819: To control the flow of complex messages through your message flows, you use event driven message processing.
- "Handling errors in message flows" on page 1883: To take a specific action in response to certain error conditions and situations, you enhance your message flows to provide your own error handling.

Message flows overview

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

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

A message flow must include an input node that provides the source of the messages that are processed. You can process the message in one or more ways, and optionally deliver it through one or more output nodes; see "Connecting client applications" on page 737. The message is received as a bit stream, and is converted by a parser into a tree structure that is used internally in the message flow. Before the message is delivered to a final destination, it is converted back into a bit stream. For more information about these conversions, see "Parsers" on page 520 and "The message tree" on page 500.

When you want to exchange messages between multiple applications, you might find that the applications do not understand or expect messages in the same format. You must provide some processing between the sending and receiving applications that ensures that both can continue to work unchanged, but can exchange messages successfully. For more information about the options available, see "Transforming and enriching messages" on page 1249.

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

The processing that you set up determines what actions are completed on a message when it is received, the order in which the actions are completed, and the final destinations of the message. All these actions manage the route that a message takes through a message flow; more information about these actions is provided in "Routing messages" on page 1233. To complete more complex processing involving more than one message, you can use the nodes described in "Processing events" on page 1819.

You can configure additional properties to make your message flow transactional, or multithreaded. You can use a global cache to share data between message flows. You can also add error paths that ensure every message is handled in an appropriate way.

When you want to run a message flow to process messages, you deploy it to an integration node, where it is run in an integration server.

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers and message flows that you can deploy. For more information, see "Restrictions that apply in each operation mode" on page 5.

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

Message flow nodes

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

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

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

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

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

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

When you develop a message flow, how you set the properties of the nodes in that flow influences how the messages are processed by that flow. For example, by setting properties that define input and output

IBM MQ queue names, you determine where the message flow receives the message from, and where it delivers the message.

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

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

Another subset of node properties is *operational properties*; that is, you can control their values by using a policy. A policy enables you to define a common approach to controlling certain aspects of message flow behavior, and particular node properties such as connection credentials. You can create and update a policy at any time in the solution lifecycle. For more information about operational policies, see “Overriding properties at run time with policies” on page 324.

The mode in which IBM App Connect Enterprise is running can affect the types of node that you can use; see “Restrictions that apply in each operation mode” on page 5.

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

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

For information about all of the built-in nodes that are supplied by IBM App Connect Enterprise, see Built-in nodes.

For information about the nodes that you can use to connect IBM App Connect Enterprise to your applications, see “Nodes for connectivity” on page 487.

User-defined node
A user-defined node is an extension to the integration node that provides a new message flow node in addition to the nodes that are supplied with the product. A user-defined node must be written to the user-defined node API provided by IBM App Connect Enterprise for both C and Java languages.

Subflow
A subflow is a directed graph that is composed of message flow nodes and connectors and is designed to be embedded in a message flow or in another subflow. To connect your subflow to other nodes in the main flow, you can add Input and Output nodes to the subflow. You can define subflows in one of two resource types, either a `.subflow` file or a `.msgflow` file. A subflow that is defined in a `.subflow` file can be deployed as an individual resource. A subflow that is defined in a `.msgflow` file must be deployed with the main flow in which it is embedded.

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

The subflow, when it is embedded in a main flow, is represented by a subflow node, which has a unique icon. The icon is displayed with the correct number of terminals to represent the Input and Output nodes that you included in the subflow definition.

The most common use of a subflow is to provide processing that is required in many places within a message flow, or is to be shared between several message flows. For example, you might code some

error processing in a subflow, or create a subflow to provide an audit trail (storing the entire message and writing a trace entry).

For more information, see "Subflows" on page 488.

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

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

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

All built-in nodes include error handling as part of their processing. If an error is detected within the node, the message is propagated to the failure terminal. What happens then depends on the structure of your message flow. You can use only the basic error handling that is provided by the integration node, or you can enhance your flow by adding error processing nodes and flows to provide more comprehensive failure processing. For more information about these options, see "Handling errors in message flows" on page 1883.

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

Message flow node palette

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

When you first open the IBM App Connect Enterprise Toolkit, the default drawers contain built-in message flow nodes that are related in function. For example, one drawer contains all the nodes that handle input and output to IBM MQ queues. Another drawer, **Transformation**, groups the nodes that you can use to convert the input message into a different form, including the Compute and JavaCompute nodes.

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

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

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

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

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

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

- "Changing the palette layout" on page 615
- "Changing the palette settings" on page 615
- "Customizing the palette" on page 616

- "Adding nodes to the Favorites category on the palette" on page 617

Nodes for connectivity

IBM App Connect Enterprise supports direct connections from applications, and can send direct requests to other application endpoints. IBM App Connect Enterprise can also connect to various subsystems including IBM MQ, files, and databases, to read and write existing application data.

You can connect IBM App Connect Enterprise to your applications by adding the appropriate nodes to your message flow. The nodes you use can be tailored to support the protocols and subsystems that your applications already use. IBM App Connect Enterprise supplies nodes to support different protocols and subsystems; you can also create your own nodes to support additional protocols and subsystems if required.

- Nodes are supplied to support the protocols that are described in "Connecting client applications" on page 737; for example, IBM MQ and HTTP.

Within your message flow, you can include the following types of nodes to communicate with your applications. The icons for the nodes in each group described are based on a common appearance, which is shown with that group.

Input nodes

The input node reads data from a subsystem or input application, which might be in the form of a message, or a record (for example, from a file). The input node calls a parser to interpret the data and create an internal message tree structure. The node can split the input message into records if required. When the message is ready, the input node sends it to the rest of the message flow for processing.

Input nodes are represented by icons that conform to this template: 

Output nodes

An output node takes data from the message tree, calls a parser to serialize the tree into the appropriate message or record format, and writes out the message or record to one or more specified end applications or subsystems. If appropriate, you can configure your message flow to continue processing a message after it has generated one or more output messages through output nodes.

Output nodes are represented by icons that conform to this template: 

Reply nodes

A reply node is a specialized form of an output node. Typically, the reply node is associated with an input node in the same flow, and uses context information from that input node to decide where to send the reply. Depending on the protocol of the input node, the context information might be created for you by that node; for other protocols, the context information might be contained within the message itself.

Reply nodes are represented by icons that conform to this template: 

Get and receive nodes

A get (receive) node reads extra data from a subsystem, and includes it in the current message tree, during message flow processing.

Get nodes are represented by icons that conform to this template: 

Request nodes

A request node writes a request to an external system, reads the response, and incorporates some or all that response data into the current message tree.

Request nodes are represented by icons that conform to this template: 

Asynchronous request and response nodes

These two nodes are specialized form of a request node, in which you can generate a request and handle the response in a second message flow. Typically, you use these nodes when you are making a request that might take some time to complete. By using this technique, you can have several outstanding requests, without suspending flow processing.

Asynchronous request nodes are represented by icons that conform to this template: 

Asynchronous response nodes are represented by icons that conform to this template: 

You can connect applications that use different protocols by choosing an appropriate mix of input and output nodes. You must also include nodes that can transform the input message into the appropriate output format between the input and output nodes.

Subflows

You define a subflow to provide a common sequence of actions to be used by several message flows, applications, or integration services. You can include subflows in your message flows in the same way as you include built-in or user-defined nodes. You can also connect subflows to other nodes in the same way.

A subflow provides the following benefits:

- Reusability and reduced development time
- Consistency and increased maintainability of your message flows (consider a subflow as analogous to a programming macro, or to inline code that is written once but used in many places)
- Flexibility to tailor a subflow to a specific context (for example, by updating the output queue or data source information)

Consider these examples of subflow use:

- You can define a subflow that provides a common sequence of actions that applies to several message flows if an error is encountered. For example, you might have a common error routine that writes the message to a database through the Database node, and puts it to a queue for processing by an error recovery routine. The use of this routine in multiple message flows, or in several places within one message flow, provides an efficient and consistent use of resources and avoids reinventing such routines every time an error is encountered.
- You might want to perform a common calculation on messages that pass through several different message flows; for example, you might access currency exchange rates from a database and apply them to calculate prices in several different currencies. You can include the currency calculator subflow in each of the message flows in which it is appropriate.

Note: You should not use a standard input node (a built-in input node such as MQInput, or a user-defined input node) in a subflow; instead use an Input node to provide the In terminal to a subflow. For more information, see node.

If you want to reuse subflows between multiple applications or integration services, store the subflows in a shared library. You can change those subflows and redeploy the shared library without the need to redploy the applications or integration services that reference the shared library.

Types of subflows

You define a subflow once, and use it in more than one message flow, application, integration service, and integration project. You define subflow content in the same way as you define message flow content, by adding, configuring, and connecting message flow nodes.

Note: At run time, each instance of a subflow creates a copy of all the message flow nodes that define that subflow. This behavior affects resource usage, which can affect your overall message flow performance.

You can create a subflow as a `.subflow` file or as a `.msgflow` file. Choose which type of subflow to use based on the following information:

A `.subflow` file subflow

A subflow that is defined in a `.subflow` file can be deployed in any of the following ways:

- The subflow is deployed separately from any of the message flows that use this subflow. The subflow and the message flows that include this subflow must be deployed in the same integration server. The subflow can be deployed directly to an integration server or as part of a library. If you update this type of subflow and redeploy it, all message flows that use this subflow, and are not part of an application or the integration service, are automatically updated. You do not need to redeploy these message flows. If you update a subflow in a shared library and redeploy it, all message flows in an application or integration service that use this subflow are updated automatically.
- The subflow is deployed as part of an application or integration service.

You cannot use the following nodes in this type of subflow:

- Nodes representing subflows that are defined in `.msgflow` files
- User-defined nodes created from subflows that are defined in `.msgflow` files

If you create a BAR file that contains both ESQL code and a subflow that is defined in a `.subflow` file, you cannot include the ESQL code directly in compiled message flow files.

A `.msgflow` file subflow

A subflow that is defined in a `.msgflow` file is embedded inside any parent message flows that use it when the message flow is placed in a BAR file. This type of subflow can only be deployed to an integration server with the message flow in which it is used. If you update this type of subflow, you must redeploy all message flows that use the subflow so that the updated subflow is used. This type of subflow can be packaged as a user-defined node.

Conditions that apply when you convert a subflow between `.msgflow` and `.subflow` and vice versa

You can convert a subflow created as a **`.msgflow`** file to a **`.subflow`** file.

- If the **`.msgflow`** file contains subflows that are defined as **`.msgflow`** files, these subflows must also be converted to **`.subflow`** files.
- If the **`.msgflow`** file is used as a subflow, the parent flow must be updated so that it references the new **`.subflow`** file.

You can convert a subflow created as a **`.subflow`** file to a **`.msgflow`** file.

- You cannot use the name of a file that already exists in that application, library, or integration project where you create your subflow as a **`.msgflow`** file. You must include **`.msgflow`** at the end of your subflow file name. An application or integration service can use subflows with the same name in one or more shared libraries if the subflows are in different broker schemas.

Conditions that apply when you want to add a subflow into a message flow

You can add subflows into a message flow if either of the following statements is true:

- The subflow that you want to add in a message flow is defined in a library, and you have specified the dependency of the current message flow container on that library. Applications and integration services can reference libraries. (A library is a logical grouping of related code, data, or both that typically contains reusable subflows, and other type of resources.)
- The subflow that you want to add in a message flow is defined in the same integration project, application, or integration service as the message flow.

When you store subflows in a shared library, you must place the subflows inside a schema that is not the default empty schema.

When an application or shared library references other shared libraries, all the subflows for a broker schema must be in a single container. Subflows for a broker schema must not be in both an application (or shared library) and a shared library that is referenced by that application (or shared library). Subflows for a broker schema must not be in two or more shared libraries that are referenced by a single application or shared library. All the subflows in a broker schema must be either in the main application or shared library, or in a single referenced shared library.

Conditions that apply when you want to add a subflow into a subflow

You can add subflows into a subflow in any of the following cases:

- You can add subflows that are defined in **.subflow** files into subflows that are defined in **.subflow** files and **.msgflow** files.
- You can add subflows that are defined in **.msgflow** files into subflows that are defined only in **.msgflow** files.
- A subflow in a shared library can contain another subflow in the same or a different shared library.

Conditions that apply when you deploy a subflow

You can deploy subflows in any of the following ways:

- Deploy a subflow as an independent resource that is defined in an integration project.
- Deploy a subflow in an application.
- Deploy a subflow as part of a service operation.
- Deploy a subflow in a library.

You deploy a subflow to an integration server by sending a BAR file to an integration server, which unpacks and stores the contents ready for when your message flows are started. You can also deploy a subflow directly to an integration server.

The following conditions apply when you deploy a subflow:

- If you deploy a message flow that contains a subflow that is defined in a **.subflow** file, the subflow is automatically included in the BAR file.
- If you deploy a subflow that is contained in an application, you must deploy the application to an integration server, which results in a complete deployment of the application.
- If you deploy a subflow that is contained in an integration service, you must deploy the service to an integration server, which results in a complete deployment of the integration service.
- If you deploy a subflow that is contained in a library, you can deploy the library to an integration server directly, outside of an application or an integration service. Then the subflow included in the library can be referenced by other message flows and subflows that are deployed directly to the same integration server as the library.
- A subflow that is created as a **.msgflow** file in an integration project can only be deployed as a separate resource if it contains at least one Input node.

Version control

Use the Passthrough node to enable version control of a subflow at run time.

By including a Passthrough node, you can add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label in a any of the following ways:

- By using the **mqsireadbar** command to read the properties stored in the BAR file.

- In the IBM App Connect Enterprise Toolkit, on the properties of a deployed message flow when it was last deployed to a particular integration node.
- In the runtime environment, if you enable user trace for that message flow.

For subflows that are created as `.subflow` files, consider the following behavior when you deploy a new version of a subflow:

- If the subflow is deployed separately from any of the message flows that use this subflow, and you deploy a new version of the subflow, all the message flows are updated automatically.
- If the subflow is deployed as part of an application or an integration service, you need to update your applications and integration services to include the new subflow version, and redeploy them.

For subflows that are created as `.msgflow` files, consider the following behavior when you deploy a new version of a subflow:

- You must update your applications, integration services, and independent resources that use the subflow to include the new subflow version, and redeploy them.

To learn more about subflows, see the following scenario Scenario: Reusing common application logic in multiple integration solution.

Callable message flows

A message flow can call another flow directly. You can deploy both flows to IBM App Connect Enterprise or IBM App Connect on IBM Cloud, or you can deploy one flow to IBM App Connect Enterprise and one flow to IBM App Connect on IBM Cloud.

A message flow can complete many different actions. If any of those actions are labor-intensive, you can split them from the main flow and complete them somewhere else. Callable flows also facilitate reuse because they can be called by multiple message flows. You can split your flows between different applications in the same integration server, or between different integrations servers, which can also be in different integration nodes. Your callable flows must be in applications. You cannot deploy callable flows in libraries or integration projects.

Some parts of a message flow might logically belong in a specific location. For example, if your flow queries an on-premises database, performance is better if that part of the flow remains on premises. But if part of your flow queries a website multiple times, performance might be improved by running that part of the flow in the cloud. You can call the cloud-based flow from your on-premises flow, and the flow in the cloud does not use any of your on-premises resources.

If your callable flows are both deployed to IBM App Connect on IBM Cloud, they can communicate with each other as soon as you deploy them. You do not need to set up communication between them. You also do not need to set up communication if both flows are deployed to the same integration server on IBM App Connect Enterprise.

If you are splitting processing between different integration servers, or between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, your flows communicate by using a *Switch server* and *connectivity agents*. The Switch server is a special type of integration server that routes data. You cannot deploy anything to the Switch server. The connectivity agents contain the certificates that your flows require to communicate securely with the Switch server. The connectivity agents must be running in the integration servers where you have deployed your on-premises message flows.

You can split processing synchronously between message flows by using the CallableFlowInvoke node in the calling flow, and CallableInput and CallableReply nodes in the callable flow. Alternatively, you can split processing asynchronously between message flows, by using the CallableFlowAsyncInvoke node in the calling flow, CallableInput and CallableReply nodes in the callable flow, and the CallableFlowAsyncResponse node in the response flow. You can also choose to share data between the flows that contain these asynchronous nodes (the calling flow and the response flow) by storing and retrieving data in the `UserContext` folder in the environment tree. For more information about sharing data between flows, see “Sharing data between a calling flow and a response flow” on page 603.

Note: This topic is about using the callable flow technique between App Connect Enterprise message flows. You can also use the callable flows technique with on-cloud event-driven flows and API flows

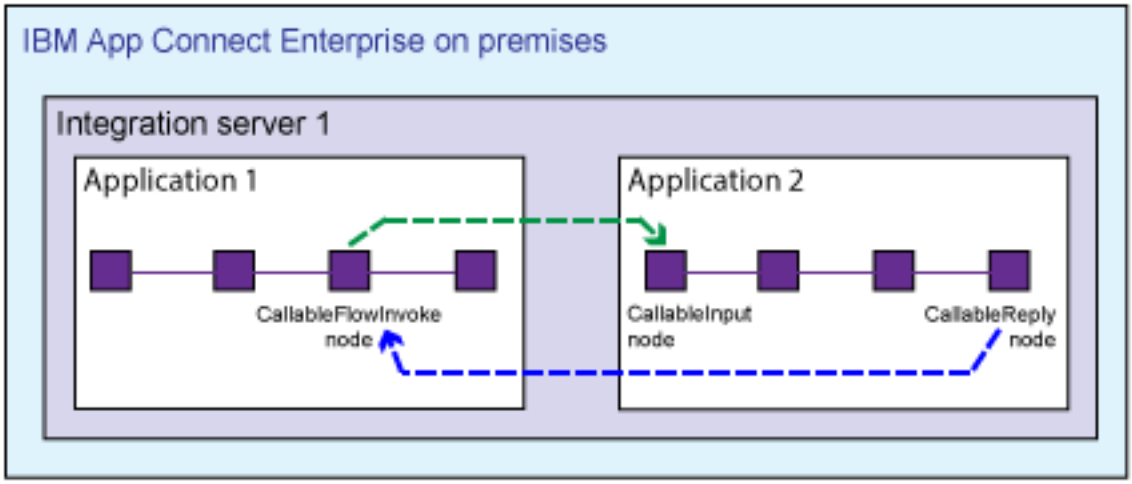
in IBM App Connect on IBM Cloud. Such on-cloud flows can be called by an App Connect Enterprise CallableFlowInvoke node, and such on-cloud flows can use the Invoke action node to call other on-cloud event-driven flows (that have the Input event node) or App Connect Enterprise message flows (that have the CallableInput node) either on cloud or on premises. For example, see the tutorial: Sharing data and processing from on-premises activities with on-cloud SaaS applications, by using an on-cloud callable flow.

You can split flows into multiple callable flows. However, for simplicity, the following scenarios assume that you are using two App Connect Enterprise message flows only: a calling flow and a callable flow.

Scenarios

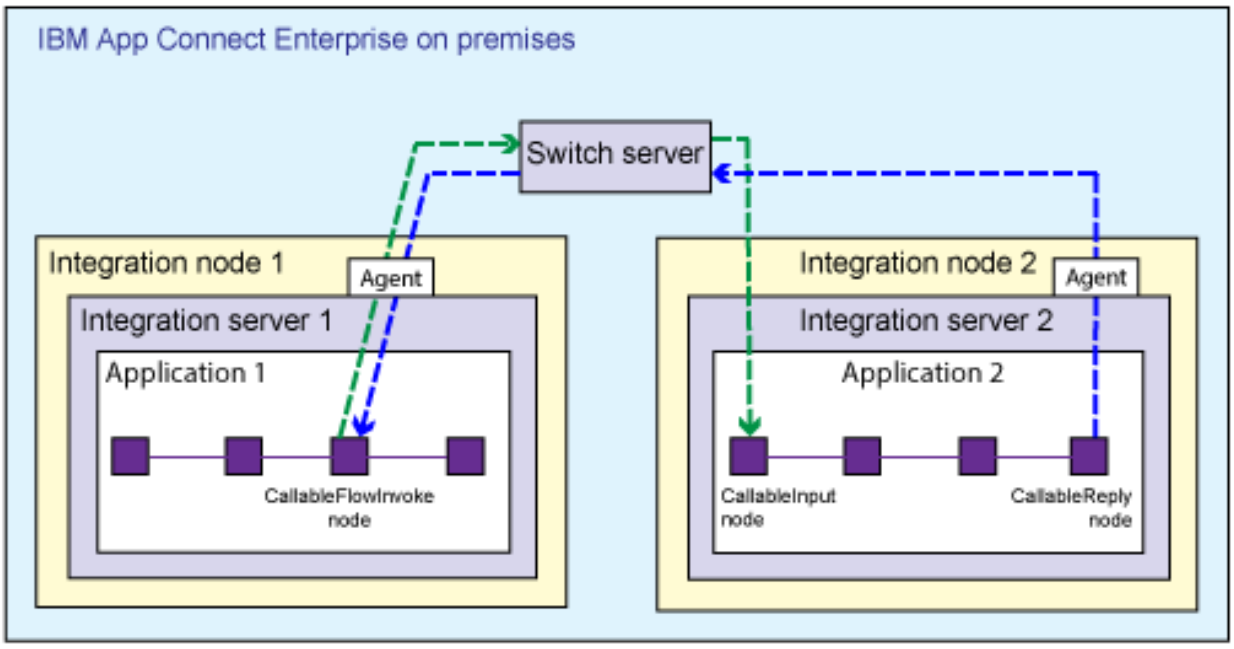
Splitting flows synchronously between applications on the same integration server

A message flow in an IBM App Connect Enterprise integration server calls a message flow in the same integration server, but in a different application. For this scenario, you do not need to set up a connection agent. This scenario is supported on all operating systems.



Splitting flows synchronously between applications on different integration servers

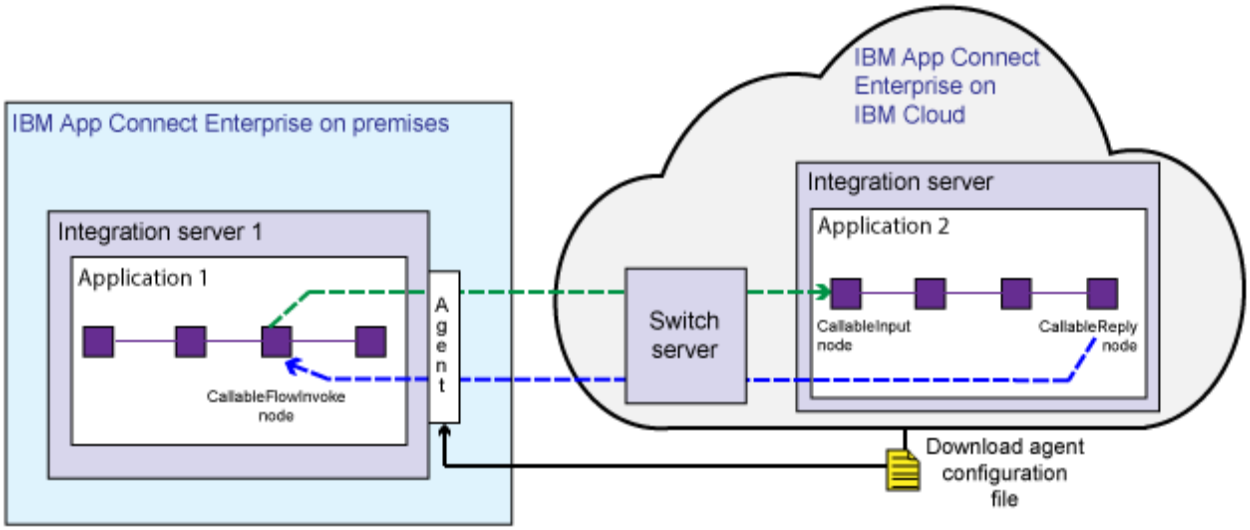
A message flow in an IBM App Connect Enterprise integration server calls a message flow in a different integration server.

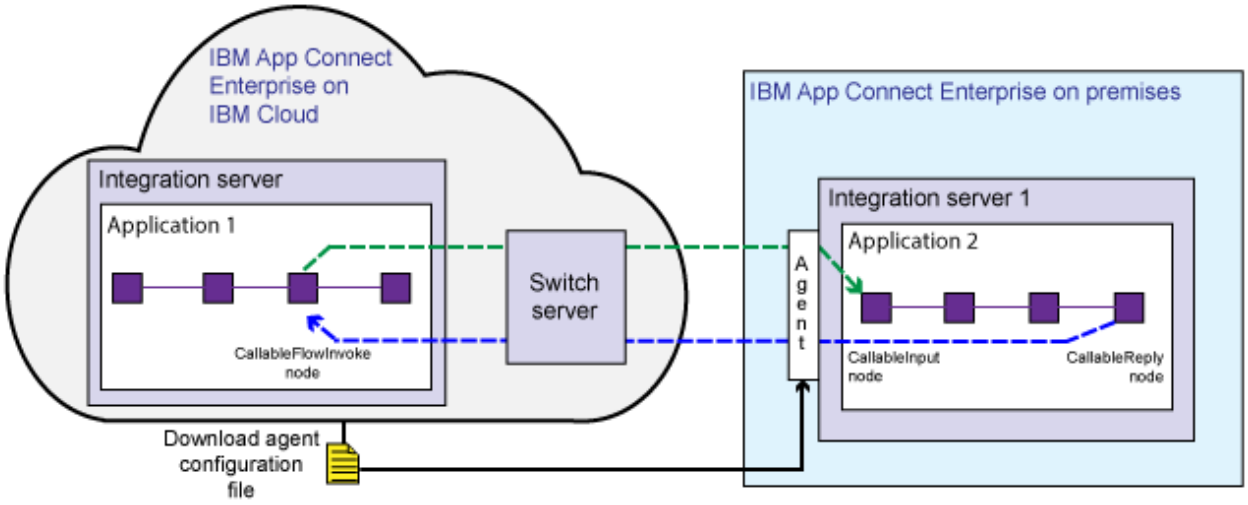


If your flows are split between different integration servers in IBM App Connect Enterprise, you must run a command that creates some configuration files. You use one of the configuration files to create the Switch server, and the other file to configure connectivity agents for each integration server. The integration servers can be in the same integration node or different integration nodes. This scenario is supported on Windows and Linux only.

Splitting flows synchronously between IBM App Connect Enterprise and IBM App Connect on IBM Cloud

A message flow in IBM App Connect Enterprise calls a message flow in IBM App Connect on IBM Cloud, or a message flow in IBM App Connect on IBM Cloud calls a message flow in IBM App Connect Enterprise.





If your flows are split between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, the Switch server is managed by IBM App Connect on IBM Cloud. You download a file from the cloud that configures an on-premises connectivity agent. This scenario is supported on Windows and Linux only.

Splitting flows asynchronously in IBM App Connect Enterprise, or between IBM App Connect Enterprise and IBM App Connect on IBM Cloud

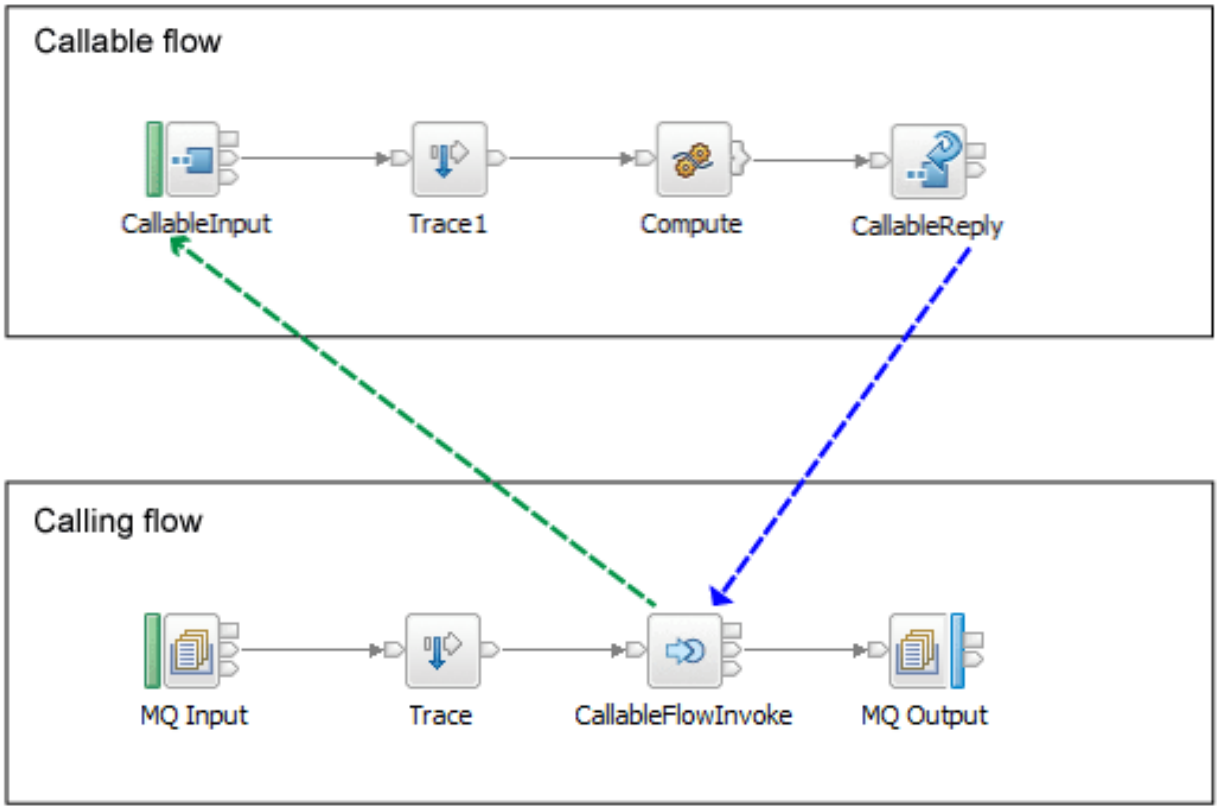
A message flow contains a CallableFlowAsyncInvoke node, which calls a second (*callable*) message flow containing a CallableInput node and a CallableReply node. A third (*response*) flow contains a CallableFlowAsyncResponse node.

When a message is passed into the calling message flow, the CallableFlowAsyncInvoke node sends the contents of the message body and environment folders to the CallableInput node of the callable flow, and then completes immediately. When the callable flow completes processing, the CallableReply node sends the message body and environment folder data to the CallableFlowAsyncResponse node in the response flow.

If your flows are split between different integration servers, you must create a Switch server (which routes data) and connectivity agents to allow the flows to communicate securely. For more information, see “Preparing the environment for callable flows” on page 222.

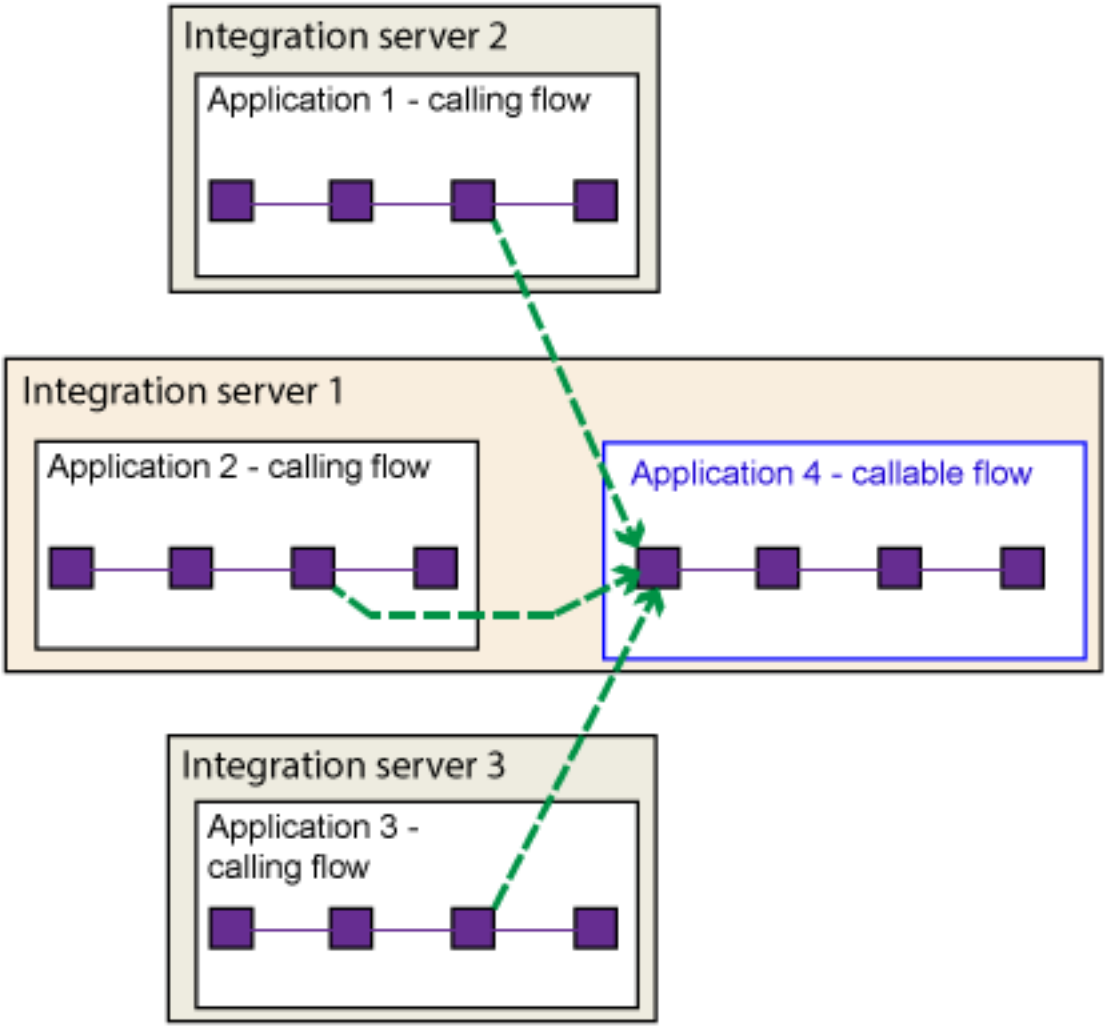
Flow configuration

When you decide how to split your message flow processing, create your message flows. The calling flow includes a CallableFlowInvoke or CallableFlowAsyncInvoke node, which calls a CallableInput node in a second (*callable*) flow. When the message is processed by the callable flow, a CallableReply node returns information to either the CallableFlowInvoke node in the calling flow (when flows that are split synchronously) or the CallableFlowAsyncResponse node (when flows that are split asynchronously). The calling flow uses a combination of the application name and the endpoint name of the CallableInput node to identify which callable flow to call.



In this example, the CallableFlowInvoke node parses the incoming message in full so that it is in a suitable format to send to the CallableInput node. Therefore, you should validate the message before it reaches the CallableFlowInvoke node. If the message fails validation at this point, it can be rolled back.

Multiple message flows can call the same callable flow. Therefore, your callable flow might complete a set of actions that are common to multiple flows.



You can add multiple CallableFlowInvoke or CallableFlowAsyncInvoke nodes to a message flow. You can also include CallableFlowInvoke or CallableFlowAsyncInvoke nodes in your callable flows. Application and endpoint name pairs must be unique on a single integration server. You can have multiple callable flows that have the same application and endpoint names, but they must be in different integration servers. In this case, the Switch server acts as a load balancer.

Deployment

After you develop your message flows, you package the containing applications into BAR files and deploy them to the appropriate locations. If the flow accesses on-premises resources, deploy its BAR file on premises. If the flow will run in the cloud, upload its BAR file to IBM App Connect on IBM Cloud.

When the calling flow receives an input message, the CallableFlowInvoke (or CallableFlowAsyncInvoke) node uses the application name and the **`Endpoint Name`** on the CallableInput node to identify and start the callable flow.

For detailed information about how to configure callable message flows, see “Splitting the processing of message flows” on page 599.

Message flow connections

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

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

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

Bend points

A bend point is added to a connection between two message flow nodes where the line that represents the connection changes direction.

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

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

Message flow node terminals

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

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

Every built-in node has a number of terminals to which you can connect other nodes. Input nodes (for example, MQInput) do not have input terminals; all other nodes have at least one input terminal through which to receive messages to be processed.

Most built-in nodes have failure terminals that you can use to manage the handling of errors in the message flow. Use the failure terminal to handle errors within the node itself.

Most built-in nodes have a catch terminal. These nodes are typically at the start of a transaction, where an uncaught exception would cause a rollback. In these nodes, the catch terminal behaves as though a TryCatch node were wired directly to the Out terminal. Use the catch terminal to handle any exceptions that are thrown beyond the message flow node.

Most nodes have output terminals through which the message can flow to a subsequent node.

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

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

Broker schemas

A broker schema is a symbol space that defines the scope of uniqueness of the names of resources defined within it. The resources include message flows and other optional resources such as ESQL files, Java files, and mapping files.

The broker schema is defined as the relative path from the project source directory to the flow name.

You can create new broker schemas to provide separate symbol spaces within the same integration project. A broker schema is implemented as a folder, or subdirectory, within the project, and provides organization within that project. You can also use project references to spread the scope of a single broker schema across multiple projects to create an application symbol space that provides a scope for all resources associated with an application suite.

If you create a new broker schema while in category mode, the empty schema is not visible in the Application Development view. To show the empty schema in the Application Development view, click **Hide Categories**  on the toolbar.

A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore. You can use the period to provide a structure to the name, for example `Stock.Common`. A directory is created in the project directory to represent the schema, and if the schema is structured using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory `Common` within a directory `Stock` within the integration project directory.

If you create a resource (for example, a message flow) in the default broker schema within a project, the file or files associated with that resource are created in the directory that represents the project. If you create a resource in another broker schema, the files are created within the schema directory.

For example, if you create a message flow in the default schema in the integration project Project1, its associated files are stored in the Project1 directory. If you create another message flow in the `Stock.Common` broker schema within the project Project1, its associated files are created in the directory `Project1\Stock\Common`.

Because each broker schema represents a unique name scope, you can create two message flows that share the same name within two broker schemas. The broker schemas ensure that these two message flows are recognized as separate resources. The two message flows, despite having the same name, are considered unique.

If you move a message flow from one project to another, you can continue to use the message flow within the original project if you preserve the broker schema. If you do this, you must update the list of dependent projects for the original project by adding the target project. If, however, you do not preserve the broker schema, the flow becomes a different flow because the schema name is part of the fully qualified message flow name, and it is no longer recognized by other projects. This action results in broken links that you must manually correct. For further information about correcting errors after moving a message flow, see “Moving a message flow or subflow” on page 578.

Do not move resources by moving their associated files in the file system; you must use the IBM App Connect Enterprise Toolkit to move resources to ensure that all references are corrected to reflect the new organization.

The following scope and reuse conditions apply when you create functions, procedures, and constants in a broker schema:

Functions

- Functions are locally reusable and can be called by module-scope subroutines or mappings within the same schema.
- Functions are globally reusable and can be called by other functions or procedures in ESQL or mapping files within any schema defined in the same or another project.

Procedures

- Procedures are locally reusable and can be called from module-scope subroutines in ESQL files within the same schema.
- Procedures are globally reusable and can be called by other functions or procedures in ESQL files within any schema defined in the same or another project.

Procedures cannot be used in mapping files.

Constants

- Constants are locally reusable and can be used where they are defined in any ESQL or mapping file within the same broker schema.
- Constants are not globally reusable; you cannot use a constant that is declared in another schema.

If you want to reuse functions or procedures globally:

- Specify the path of the function or procedure:
  - If you want to reuse a function or procedure in an ESQL file, either provide a fully-qualified reference, or include a PATH statement that defines the path.

    If you define the path, code the PATH statement in the same ESQL file as that in which the function is coded, but not within any MODULE.
  - If you want to reuse a function in a mapping file, take one of the following steps:
    - Qualify the function in the Composition Expression editor.
    - Select **Organize Schema References** in the outline view. This detects dependent PATHs and automatically updates the reference.
    - Select **Modify Schema References** in the outline view. You can then select the schema in which the function is defined.

    (You cannot reuse a procedure in a mapping file.)
- Set up references between the projects in which the functions and procedures are defined and used.

Client application programming interfaces

You can configure the nodes in your message flows to customize the behavior of those nodes by using one or more of the supported programming interfaces.

You can use one of the following options for converting message formats in your message flows:

Mappings

The Graphical Data Mapping editor in the IBM App Connect Enterprise Toolkit is a graphical interface that displays a visual representation of messages. You can use this editor to:

- Drag fields from a source message to a target message
- Map data from databases to the message structure
- Apply functions

You can use mappings in the Mapping node.

ESQL

Use ESQL to manipulate data in both messages and database tables. ESQL is a programming language that is specific to IBM App Connect Enterprise, and based on SQL. You can code ESQL statements to create, reference, and update message fields and database content. ESQL provides a rich set of statements and functions that you can use to achieve sophisticated transformations.

You can use ESQL in the Compute, Database, and Filter nodes.

Java

Use the Java programming language to route or transform your messages. You can use XPath to create, reference, and update message fields. You can also use JDBC to access database tables.

You can use Java only in the JavaCompute node.

XSL style sheets
Use standard XSL style sheets to convert XML messages to other formats supported by the integration node.

You can use XSL only in theXSLTransform node.

Use this option if your message flows are processing XML messages, and your message flow designers are familiar with the XSL style sheets.

Windows .NET
Use the .NET framework to route or transform your messages. You can use .NET to create, reference, and update message fields.

You can use .NET in the .NETCompute node, and call .NET code from ESQL.

When you configure your message flows and nodes, you create a set of files that are stored in your workspace.

The files created are of the following types:

- A message flow definition file, *message_flow_name*`.msgflow`. This file is mandatory, and is created automatically for you. It contains details about the message flow characteristics and contents; for example, what nodes it includes and its promoted properties.
- One or more message mappings files, *message_flow_name_nodename*`.map`. A unique file is required for each node in the message flow that uses the Graphical Data Mapping editor. This file is required only if your message flow contains a Mapping node. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.
- One or more ESQL resources files, *message_flow_name*`.esql`. An ESQL file is required only if your message flow includes one or more of the nodes that must be customized by using ESQL modules, or contains functions that are called by your message mappings. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

  You can customize the following built-in nodes by creating free-form ESQL statements that use the built-in ESQL statements and functions, and your own user-defined functions:

  – Compute
  – Database
  – Filter
- One or more Java programming files, *message_flow_name*`.java`. A Java file is required only if your message flow includes one or more JavaCompute nodes. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.
- One or more XSL stylesheets, *message_flow_name*`.xslt`. An XSLT file is required only if your message flow includes one or more XSLTransform nodes. You can create this file yourself, or you can cause it to be created for you by requesting specific actions against a node.

You can include other files in your integration project so that they are deployed to the integration node with your message flow. The integration node stores these extra files but does not process them in any way.

For details of how to create the files to support these transform options, and create their content, see “Transforming and enriching messages” on page 1249.

The message tree

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

A message is used to describe:

- A set of business data that is exchanged by applications

- A set of elements that are arranged in a predefined structure
- A structured sequence of bytes

IBM App Connect Enterprise routes and manipulates messages after converting them into a logical tree. The process of conversion, called parsing, makes obvious the content and structure of a message, and simplifies later operations. After the message has been processed, the parser converts it back into a bit stream.

IBM App Connect Enterprise supplies a range of parsers to handle the many different messaging standards in use.

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

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

Logical tree (message assembly)

The logical tree is the internal representation of a message, and is also known as the message assembly.

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

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

Several interfaces are provided:

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

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

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

- “Message tree” on page 502
- “Environment tree” on page 508
- “Local environment tree” on page 509
- “Exception list tree” on page 516

The first of these trees is populated with the contents of the input message bit stream. The remaining three trees are initially empty.

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

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

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

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

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

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

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

  You can also code ESQL to populate the output message, Environment, LocalEnvironment, or ExceptionList tree with information that is not copied from the input tree. For example, you can retrieve data from a database, or calculate content from the input message data.
- You can produce similar results in the JavaCompute node. See “Writing Java” on page 1755 for more information.

If messages are recorded as they pass through a message flow, you can view and edit the stored message assembly files by using the Message Assembly editor. You can also use the recorded message assemblies in unit tests for your message flows and message flow nodes. For more information, see “Developing integration tests” on page 1891 and #unique_720.

Message tree
The message tree is a part of the logical tree (message assembly) in which the internal representation of the message body is stored.

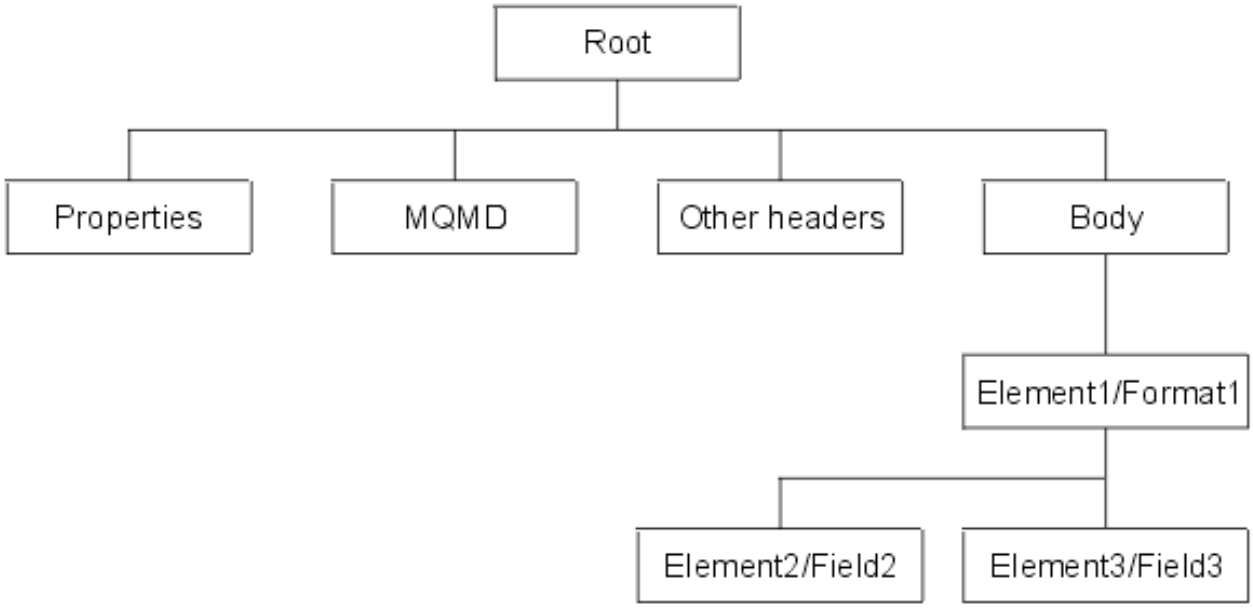
The root of a message tree is called Root. The message tree is always present, and is passed from node to node in a single instance of a message flow.

The message tree includes all the headers that are present in the message, in addition to the message body. The tree also includes the Properties subtree, if that is created by the parser. If a supplied parser has created the message tree, the element that represents the Properties subtree is followed by zero or more headers.

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

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

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



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

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

Name element
A name element has a string associated with it, which is the name of the element. An example of a name element is `XMLElement`. A name element also has a second string associated with it, which is the namespace of the element; this string might be empty.

Value element
A value element has a value associated with it. An example of a value element is `XMLContent`.

Name-value element
A name-value element is an optimization of the case where a name element contains only a value element and nothing else. The element contains both a name and a value. An example of a name-value element is `XMLAttribute`.

Properties folder

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

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

The Properties folder contains a set of standard properties that you can manipulate in the message flow nodes in the same way as any other property. Some of these fields map to fields in the supported IBM MQ headers, if present, and are passed to the appropriate parser when a message is delivered from one node to another.

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

The Properties parser ensures that the values in the header fields match the values in the Properties folder on input to, and output from, every node. For any field, if only one header is changed (the Properties

header or a specific message header), that value is used. If both the Properties header and the specific message header are changed, the value from the Properties folder is used.

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

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

When the input node receives the input message, it creates and completes the Properties tree (the first subtree of the message tree).

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

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

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

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

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

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

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

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

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

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

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

The **CodedCharSetId** and **Encoding** fields are not populated in the same way as the **Format** field. In particular, the message body is not used to determine the **CodedCharSetId** and **Encoding** values. Rather, these values affect the way in which the message body is written. This can cause unexpected results if an intermediate header (for example MQRFH2) is removed without updating the preceding header in the chain with the removed header values.

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

- If the message has an MQRFH or MQRFH2 header, the domain that is identified in the header (either in **Format** or the name and value data) determines the parser that is associated with this message.
- If the message does not have an MQRFH or MQRFH2 header, or if the header does not identify the domain, the *Message Domain* property of the input node indicates the domain of the message, and the parser that is to be used. You can specify a user-defined domain and parser.
- If the message domain cannot be identified by header values or by the *Message Domain* property of the input node, the message is handled as a binary object (BLOB). The BLOB parser is associated with the message. A BLOB can be interpreted as a string of hexadecimal characters, and can be modified or examined in the message flow by specifying the location of the subset of the string.

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

For example, the message body is parsed when you refer to a field in the message body, for example: `Root.XMLNSC.MyDoc.MyField`. Depending on the paths that are taken in the message flow, this can take place at different points. This parsing when first needed approach is also referred to as "partial parsing" or "on-demand parsing", and in typical processing does not affect the logic of a message flow. However, there are some implications for error handling scenarios; see “Handling errors in message flows” on page 1883.

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

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

WebSphere Broker File Transport, WebSphere Broker Adapters Transport, IBM MQ Web Services Transport, and WebSphere Broker JMS Transport protocols

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

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

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

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

All other protocols

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

This interface does not automatically generate a Properties subtree for a message. A message does not need to have a Properties subtree, but you might find it useful to create one to provide a consistent message tree structure, regardless of input node. If you are using a user-defined input node, you must create a Properties subtree in the message tree yourself.

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

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

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

Properties versus MQMD folder behavior for various transports

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

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

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

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

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

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

The behavior is not unique to just the **CorrelId** to **ReplyIdentifier** fields, but applies to the following fields between the MQMD and Properties folder:

- **CorrelId**
- **Encoding**

- **CodedCharSetId** The `Properties.CodedCharSetId` represents the CodedCharSetId for the message payload, which matches the CodedCharSetId in any message header that immediately precedes the message payload.
- **Persistence**
- **Expiry**
- **Priority**

In summary:

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

Example

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

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

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

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

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

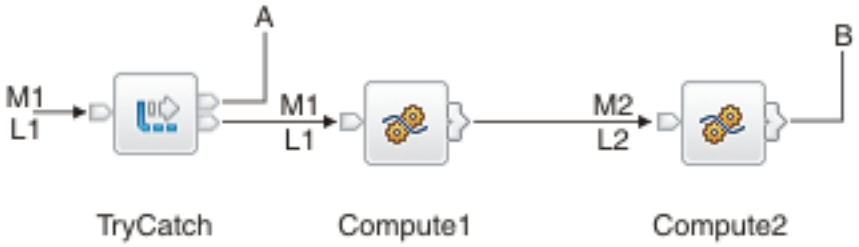
- If the exception is returned to the input node, and the input node Catch terminal is not connected, the trees are discarded. If the message is in a transaction, it is returned to the input queue for further processing. When the message is processed again, a new tree structure is created. If the message is not in a transaction, it is discarded.
- If the exception is returned to the input node and the Catch terminal is connected, the message and local environment trees that were created originally by the input node, and propagated through the Out terminal, are restored, and any updates that you made to their content in the nodes that followed the input node are lost. The environment tree is not restored, and its contents are preserved. If the nodes following the input node include a Compute node that creates a new local environment or message tree, those trees are lost. The exception list tree reflects the one or more exceptions that have been recorded.
- If the exception is caught in the message flow by a TryCatch node, the message and local environment trees that were previously propagated through the Try terminal of the TryCatch node are restored

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

Exception handling paths in a message flow

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

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



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

Environment tree

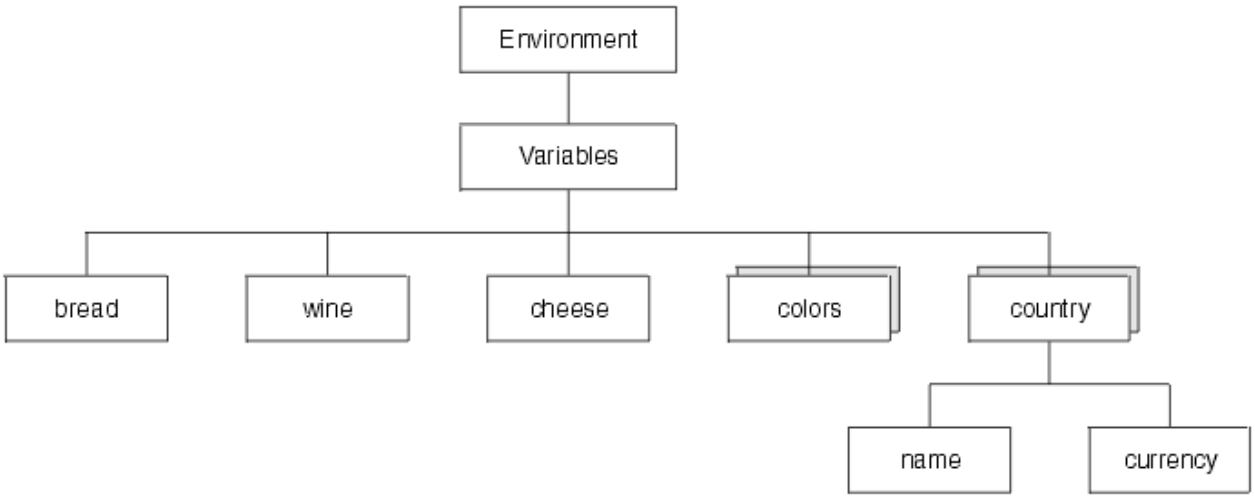
The environment tree is a part of the logical tree (message assembly) in which you can store information while the message passes through the message flow.

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

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

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

The following figure shown an example of an environment tree:



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

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

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

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

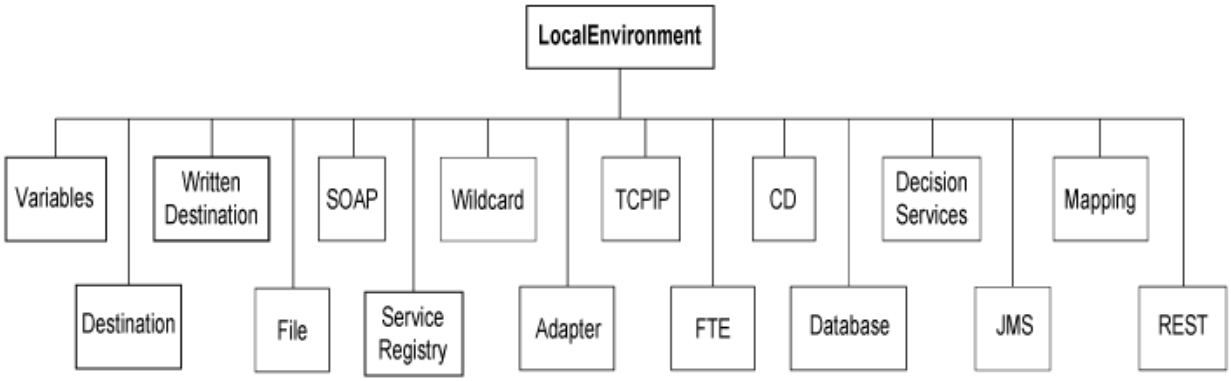
The root of the local environment tree is called `LocalEnvironment`. This tree is always present in the input message, and is created when a message is received by the input node. Some input nodes create local environment fields, other nodes leave the tree empty.

The local environment tree is made up of the following structure:

- Anything in the format of `LocalEnvironment.Destination.`*`output_or_request_node_name`*, for example; LocalEnvironment.Destination.Email, decides what happens when information is going into an output or request node.
- Anything in the format of `LocalEnvironment.WrittenDestination.`*`output_or_request_node_name`*, for example; LocalEnvironment.WrittenDestination.FTE, gives you information about the processed output of an output or a request node.
- Anything in the format of `LocalEnvironment.`*`input_node_name`*`.input`, for example; LocalEnvironment.Adapter.Input, contains information that has been stored by an input node.

Use the local environment tree to store variables that can be referred to and updated by message processing nodes that occur later in the message flow. You can also use the local environment tree to define destinations (that are internal and external to the message flow) to which a message is sent. IBM App Connect Enterprise also stores information in the local environment in some circumstances, and references it to access values that you might have set for destinations.

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



In the tree structure shown, LocalEnvironment has several children:

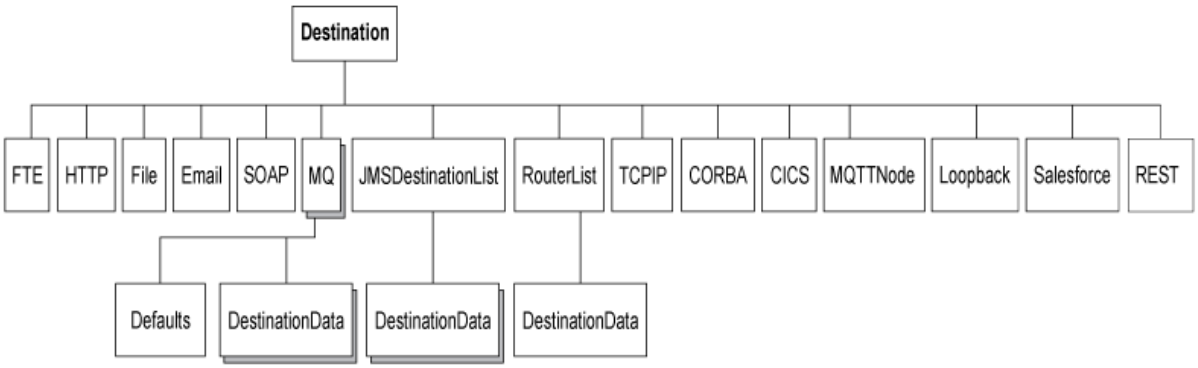
LocalEnvironment.Variables

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

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

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

LocalEnvironment.Destination



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

- Transport information

  Transport information is used by some input and output nodes, including FTE, HTTP, File, Email, SOAP, MQ, JMS, TCPIP, CORBA, CICS, and MQTT.

  LocalEnvironment.Destination.FTE

  If the message flow includes an FTEOutput node, you can override its properties with elements in this subtree. See “Using local environment variables with file nodes” on page 962.

  LocalEnvironment.Destination.HTTP

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

information, see *Local environment overrides* in the topic node, and examples in "Populating Destination in the local environment tree" on page 1656.

LocalEnvironment.Destination.File

If the message flow includes a FileOutput node, you can override its `directory` and `name` properties with elements in this subtree. See "Using local environment variables with file nodes" on page 962.

LocalEnvironment.Destination.Email

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

LocalEnvironment.Destination.SOAP

You can place outbound WS-Addressing header information in the local environment to override the defaults that are generated by the SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes. See "WS-Addressing information in the local environment" on page 834.

If the message flow includes SOAPRequest or SOAPAsyncRequest nodes, you can override their HTTP Transport and JMS transport properties in this subtree. See node, node, or Local environment overrides for the node.

If the message flow includes a SOAPAsyncRequest node, you can use this subtree to pass state and correlation information to a SOAPAsyncResponse node in another message flow. See "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 833.

If the message flow includes SOAPReply, SOAPRequest, or SOAPAsyncRequest nodes, you can override their use of outbound MTOM messages in this subtree. See "SOAP MTOM and the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 849.

LocalEnvironment.Destination.MQ

If the message flow includes an MQOutput node, each element is a name element, *MQ* (A deprecated alternative exists, called *MQDestinationList*. Use *MQ* for all new message flows). If more than one element exists, each is processed sequentially by the node. See the example in "Populating Destination in the local environment tree" on page 1656.

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

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

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

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

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

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

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

The information that you insert into DestinationData depends on the characteristic of the corresponding node property: This information is described in “Accessing the local environment tree” on page 1654.

LocalEnvironment.Destination.JMSDestinationList

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

The JMSOutput node searches the local environment for data elements called DestinationData under the folder Destination.JMSDestinationList. The node sends the JMS message to each

DestinationData entry found in that folder. See the example in “Populating Destination in the local environment tree” on page 1656.

LocalEnvironment.Destination.TCPIP

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

LocalEnvironment.Destination.CORBA

If the message flow includes a CORBARequest node, you can override its `Operation name` property by specifying a value in the following location:

```
$LocalEnvironment/Destination/CORBA/Request/OperationName
```

For more information about the operation name, see node.

LocalEnvironment.Destination.CICS

If the message flow includes a CICSRequest node, you can override the following properties with elements in this subtree:

- `Program name`
- `Commarea length`
- `Mirror transaction ID`
- `Set EIBTRNID only`
- `Message domain`
- `Message set`
- `Message type`
- `Message format`
- `Message coded character set ID`
- `Message encoding`

For more information, see “Local environment overrides for the CICSRequest node” on page 1185.

You can also set LocalEnvironment values for CICS channels and containers. For more information, see “COMMAREA or channel data structures” on page 1171.

LocalEnvironment.Destination.MQTT.Output

If the message flow includes an MQTTPublish node, you can override the following properties with elements in this subtree:

- `Retained`
- `Topic name`
- `Quality of service`

For more information, see “Using local environment variables with MQTT nodes” on page 890.

LocalEnvironment.Destination.Loopback

If the message flow includes a LoopBackRequest node, you can override the properties of those nodes with elements in this subtree. For more information, see “Using local environment variables with LoopBackRequest nodes” on page 1226.

LocalEnvironment.Destination.Salesforce

If the message flow includes a SalesforceRequest node, you can override the properties of those nodes with elements in this subtree. For more information, see “Using local environment variables with Salesforce nodes” on page 1211.

LocalEnvironment.Destination.REST

If the message flow includes a RESTRequest or RESTAsyncRequest node, you can override the properties of those nodes with elements in this subtree. For more information, see "Using local environment variables with REST nodes" on page 1039.

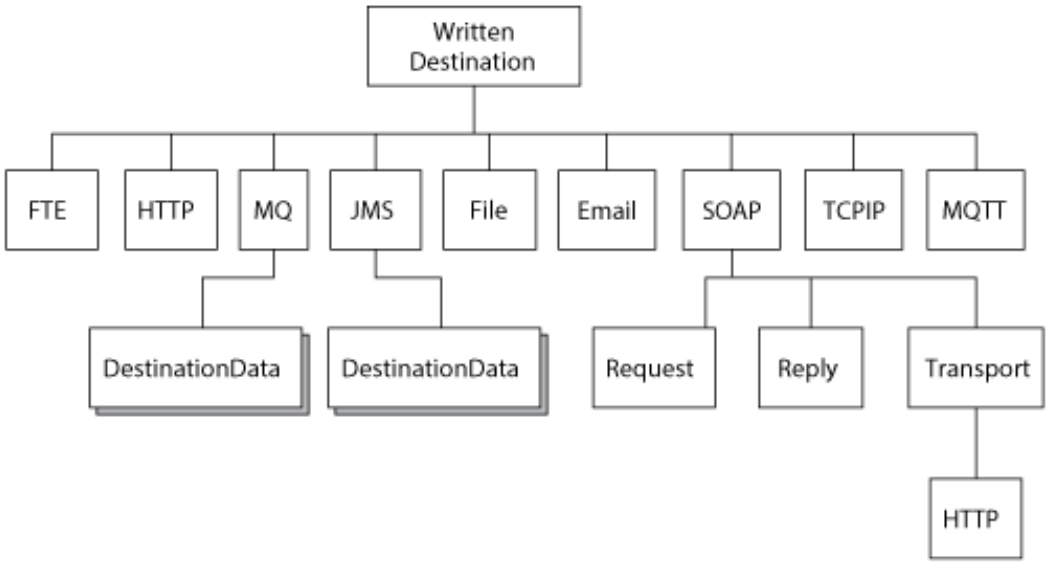
LocalEnvironment.Destination.Kafka

If the message flow includes a KafkaProducer node, you can override the properties of the node with elements in this subtree. For more information, see "Using local environment variables with Kafka nodes" on page 907.

- Routing information

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

LocalEnvironment.WrittenDestination



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

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

- Connect a transformation node to the Out terminal.

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

LocalEnvironment.File

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

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

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 962.

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

More details about the information that is stored in this subtree are in "WS-Addressing information in the local environment" on page 834.

If the message flow includes a SOAPAsyncResponse node, you can use this subtree to receive state and correlation information passed by a SOAPAsyncRequest node in another message flow.

More details about the information that is stored in this subtree are in "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 833.

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

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

More details about the information that is stored in this subtree are in "Using local environment variables with file nodes" on page 962.

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

For a WebSphere Adapters input node:

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

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

For a WebSphere Adapters request node:

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

LocalEnvironment.TCPIP

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

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

This information describes the connection that the node is using.

More details about the information that is stored in this subtree are in node, node, node, and node.

LocalEnvironment.FTE and LocalEnvironment.FTE.Transfer
These subtrees contain information that is stored by the FTEInput node. The LocalEnvironment.FTE subtree contains information about the current record. The LocalEnvironment.FTE.Transfer subtree contains information received from IBM MQ File Transfer Edition regarding the file.

More details about the information that is stored in these subtrees are in "Using local environment variables with file nodes" on page 962.

LocalEnvironment.CD and LocalEnvironment.CD.Transfer
These subtrees contain information that is stored by the CDInput node. The LocalEnvironment.CD subtree contains information about the current record. The LocalEnvironment.CD.Transfer subtree contains information received from IBM Sterling Connect:Direct regarding the file.

More details about the information that is stored in these subtrees are in "Using local environment variables with file nodes" on page 962.

LocalEnvironment.Database
This subtree contains information that is propagated from the DatabaseInput node.

The LocalEnvironment.Database.Input.Event.Usr subtree contains user-defined data associated with an event. It is initialized in the `ReadEvents` procedure of the ESQL module associated with the DatabaseInput node.

LocalEnvironment.Database.Input.Event.Key holds a unique key for an event. It is set in the `ReadEvents` procedure of the ESQL module associated with the DatabaseInput node.

LocalEnvironment.Database.Input.Event.FailureCount contains a value for the number of times an attempt to process an event has failed. This count includes all unhandled exceptions occurring either in the ESQL module or the message flow.

LocalEnvironment.JMS
This subtree contains information that is stored by the JMSReceive node.

If the message flow includes a JMSReceive node, you can override its JMS connection properties with elements in this subtree.

More details about this information that is stored in this subtree are in Local environment overrides for the node.

LocalEnvironment.Mapping
This subtree contains information that is stored by Mapping node.

If your message flow includes a Mapping node, you can override the mapping routine that is used to transform a message instance by specifying a new mapping routine in the **MappingRoutine** field. You must specify the new mapping routine in the LocalEnvironment.Mapping subtree that is upstream of the Mapping node that you need to modify.

LocalEnvironment.REST

This subtree contains information that is stored by a REST API when it receives a request and routes that request to a subflow that implements an operation in the REST API. For more information, see "Implementing REST API operation processing in the subflow by using message flow nodes" on page 2031.

This subtree also contains information that is stored by a RESTAsyncResponse node when it receives a response for a request initiated by a RESTAsyncRequest node. For more information, see "Using local environment variables with REST nodes" on page 1039.

LocalEnvironment.Kafka

This subtree contains information that is stored by the KafkaConsumer node, including information about the topic where the message was published. For more details about the information contained in this subtree, see "Using local environment variables with Kafka nodes" on page 907.

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

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

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

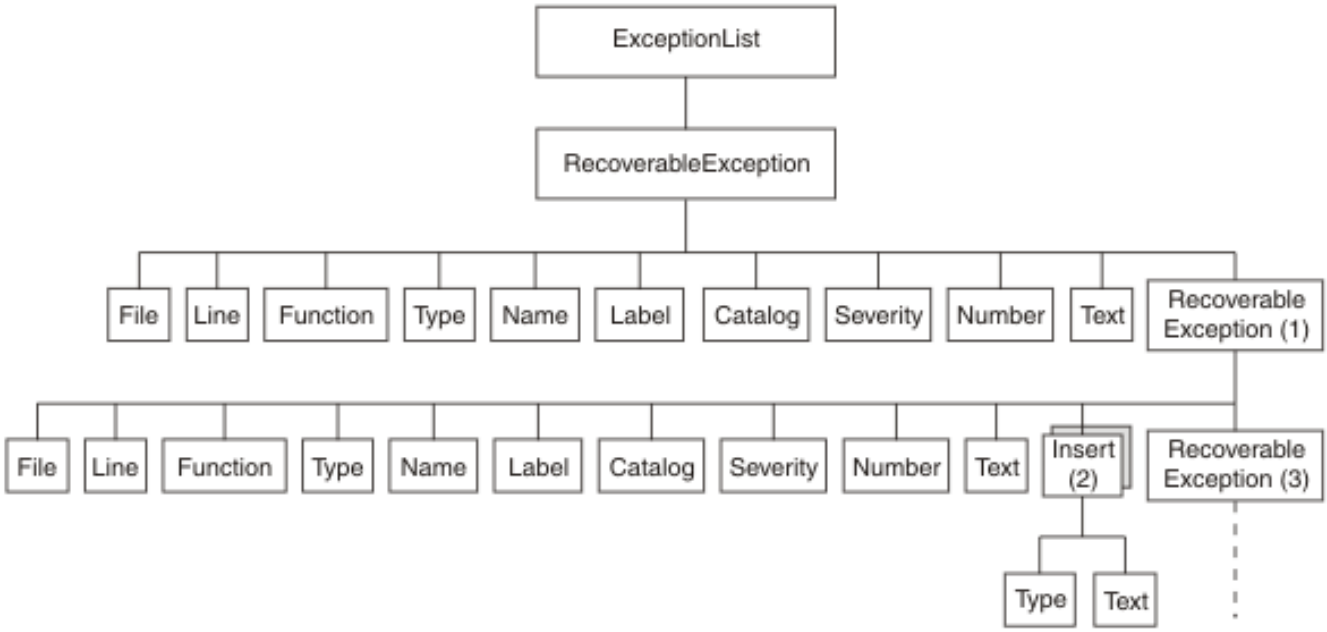
The exception list tree can be accessed by other nodes in the message flow that receive the message after the exception has occurred. You can modify the contents of the exception list tree only in a node that provides an interface to modify the outbound message tree; for example, the Compute node.

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

The child of ExceptionList may be any one of the exception types that is included in the following list. Typically, only one child of the root is created, although more than one might be generated in some circumstances. The child of ExceptionList contains a number of children, each of which may also be any one of the types in the following list. The last of these children provides further information specific to the type of exception.

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

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



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

- The depth (that is, the number of parent-child steps from the root) represents increasingly detailed information for the same exception.

- The width of the tree represents the number of separate exception conditions that occurred before processing was abandoned. This number is usually one, and results in an exception list tree that consists of a number of exception descriptions that are connected as children of each other.
- At the numbered points in the tree:
  1. This child can be any one of the exception types that are listed earlier in this topic. All of these elements have the children shown; if present, the last child is the same element as its parent.
  2. This element might be repeated.
  3. If present, this child contains the same children as its parent.

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

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

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

Notes:

1. Do not use the File, Line, Function, and Text elements for exception handling decision making. These elements ensure that information can be written to a log for use by IBM Service personnel, and are subject to change in both content and order.
2. The Type, Name, and Label elements define the object (usually a message flow node) that was processing the message when the exception condition occurred.
3. The Catalog, Severity, and Number elements define an NLS message: the Insert elements that contain the two name-value elements shown define the inserts into that NLS message.
4. NLS message catalog name and NLS message number refer to a translatable message catalog and message number.

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

Correlation names

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

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

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

Correlation names in nodes that do not create an output message

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

Root
The root of the message passing through the node.

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

Properties
The standard properties of the input message.

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

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

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

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

You cannot use these correlation names in the expression of any mapping for a Mapping node.

Correlation names in nodes that create an output message

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

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

InputRoot
The root of the input message.

InputProperties
The standard properties of the input message.

Environment
The structure that contains the current global environment variables that are available to the node. Environment can be read and updated.

InputLocalEnvironment
The structure that contains the local environment variables for the message passing through the node.

InputDestinationList
The structure that contains the local environment variables for the message passing through the node. Use the correlation name InputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name InputLocalEnvironment

InputExceptionList
The structure that contains the exception list for the message passing through the node.

OutputRoot
The root of the output message.

In a Compute node, the correlation name OutputBody is not valid.

OutputLocalEnvironment
The structure that contains the local environment variables that are sent out from the node.

While this correlation name is always valid, it has meaning only when the `Compute Mode` property of the Compute node indicates that the Compute node is propagating the LocalEnvironment.

OutputDestinationList
The structure that contains the local environment variables that are sent out from the node. Use the correlation name OutputDestinationList for compatibility with earlier versions; if compatibility is not required, use the preferred name OutputLocalEnvironment

OutputExceptionList
The structure that contains the exception list that the node is generating.

While this correlation name is always valid, it has meaning only when the `Compute Mode` property of the Compute node indicates that the Compute node is propagating the ExceptionList.

For a description of how to use *, see “Using anonymous field references” on page 1639.

Parsers

A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

A parser is called when the bit stream that represents an input message is converted to the internal form that can be handled by the integration node; this invocation of the parser is known as *parsing*. The internal form, a logical tree structure, is described in “Logical tree (message assembly)” on page 501. It is described as a tree because messages are typically hierarchical in structure; a good example of this structure is XML. The way in which the parser interprets the bit stream is unique to that parser; therefore, the logical message tree that is created from the bit stream varies from parser to parser.

The parser that is called depends on the structure of a message, referred to as the *message template*. Message template information comprises the message domain, message model, message type, and physical format of the message. Together, these values identify the structure of the data that the message contains.

A parser is also called when a logical tree that represents an output message is converted into a bit stream; this action by the parser is known as *writing*. Typically, an output message is generated by an output node at the end of the message flow. However, you can connect more nodes to an output node to continue processing of the message.

The message domain identifies the parser that is used to parse and write instances of the message. The remaining parts of the message template, message model, message type, and physical format, are optional, and are used by model-driven parsers such as the MRM parser.

The logical structure of the message typically maps to the business content of the message; for example, it contains a customer name, address, and account number. It is only when you send a message across a connection that the physical characteristics are important, and influence the construction of the bit stream.

The integration node requires access to a parser for every message domain to which your input messages and output messages belong. In addition, the integration node requires a parser for every identifiable message header that is included in the input or output message. Parsers are called when required by the message flow.

Body parsers

IBM App Connect Enterprise provides built-in support for messages in the following message domains by providing message body parsers:

- MRM (“MRM parser and domain” on page 545)
- XMLNSC, XMLNS, and XML (“XML parsers and domains” on page 528)
- SOAP (“SOAP parser and domain” on page 526)
- DataObject (“DataObject parser and domain” on page 546)
- JMSMap and JMSStream (“JMS parsers and domains” on page 547)
- MIME (“MIME parser and domain” on page 547)
- BLOB (“BLOB parser and domain” on page 552)
- IDOC (“IDOC parser and domain” on page 553)
- JSON (“JSON parser and domain” on page 553)
- DFDL (“DFDL parser and domain” on page 544)

See “Which body parser should you use?” on page 524 for a discussion about which message body parser to use under what circumstances.

You specify which message domain to use for your message at the place in the message flow where parsing or writing is initiated.

- To parse a message bit stream, typically you set the *Message Domain* property of the input node that receives the message. But, if you are initiating the parse operation in ESQL, use the **DOMAIN** clause of the **CREATE** statement.

  The message tree that is created is described in “Message tree” on page 502. Its exact form might change as it progresses through the message flow, depending on what the nodes are doing.

  The last child element of the Root element of the message tree takes the name of the body parser that created the tree. For example, if the *Message Domain* property was set to `MRM`, the last child element of Root is called MRM, which indicates that the message tree is owned by the MRM parser.
- To write a message, the integration node calls the owning body parser to create the message bit stream from the message tree.

Some body parsers are *model-driven*, which means that they use predefined messages from a message set when parsing and writing. The MRM, SOAP, DataObject, IDOC, and (optionally) XMLNSC parsers are model-driven parsers. To use these parsers, messages must be modeled in a message set and deployed to the integration node from the IBM App Connect Enterprise Toolkit.

Other body parsers are *programmatic*, which means that the messages that they parse and write are *self-defining* messages, and no message set is required. See "Predefined and self-defining messages" on page 523.

When you use a model-driven parser, you must also specify the message model and, optionally, the message type and message format so that the parser can locate the deployed message definition with which to guide the parsing or writing of the message.

To parse a message bit stream, typically you set the `Message model`, `Message `, and `Physical format` properties of the input node that receives the message. Or, if you are initiating the parse operation in ESQL, you use the **SETTYPE**, and **FORMAT** clauses of the **CREATE** statement. This information is copied into the *Properties* folder of the message tree.

To write a message, the integration node calls the owning body parser to create the message bit stream from the message tree. If the parser is a model-driven parser, it uses the *MessageSet*, *MessageType*, and *MessageFormat* fields in the Properties folder.

Whether the message type or message format are needed depends on the message domain.

Even if the body parser is not model-driven, it is good practice to create and use a message set in the IBM App Connect Enterprise Toolkit, because it simplifies the development of your message flow applications, even though the message set is not deployed in the IBM App Connect Enterprise runtime environment. See "Why model messages?" on page 2138 for information about the advantages of creating a message set.

Header parsers

IBM App Connect Enterprise also provides parsers for the following message headers, which your applications can include in input or output messages:

- WMQ MQMD (The MQMD parser)
- WMQ MQMDE (The MQMDE parser)
- WMQ MQCFH (The MQCFH parser)
- WMQ MQCIH (The MQCIH parser)
- WMQ MQDLH (The MQDLH parser)
- WMQ MQIIH (The MQIIH parser)
- WMQ MQRFH (The MQRFH parser)
- WMQ MQRFH2 and MQRFH2C (The MQRFH2 and MQRFH2C parsers)
- WMQ MQRMH (The MQRMH parser)
- WMQ MQSAPH (The MQSAPH parser)
- WMQ MQWIH (The MQWIH parser)
- WMQ SMQ_BMH (The SMQ_BMH parser)
- JMS header ("Representation of messages in the JMS Transport" on page 858)
- HTTP headers ("HTTP headers" on page 791)

All header parsers are programmatic and do not use a message set when parsing or writing.

User-defined parsers

To parse or write message body data or headers that the supplied parsers do not handle, you can create user-defined parsers that use the IBM App Connect Enterprise user-defined parser programming interface.

Tip: No parser is provided for messages, or parts of messages, in the WMQ format `MQFMT_IMS_VAR_STRING`. Data in this format is often preceded by an MQIIH header (format `MQFMT_IMS`). IBM App Connect Enterprise treats such data as a BLOB message. If you change the CodedCharSetId or the encoding of such a message in a message flow, the `MQFMT_IMS_VAR_STRING` data is not converted, and the message descriptor or preceding header does not correctly describe that part of the message. If you need the data in these messages to be converted, use the MRM domain and create a message set to model the message content, or provide a user-defined parser.

Root parsers

A root parser is the first parser in the Logical Tree structure built by the integration node. The root parser is defined by which input node you are using in your message flow, for example theSOAPInput Node uses a different root parser to the MQInput Node. Each root parser creates different properties parsers to work with the properties folder in the Logical Tree because different input nodes need to obtain and serialize these values in transport specific ways. The root parser that is assigned to different trees can cause different behavior when elements are copied between trees. The IBM MQ Root parser is used when you create new trees in ESQL.

Parser Copies

When you copy a tree between one parser and another, for example as a result of using ESQL then the behavior is different depending on the types of the parsers that are involved.

Like Parser Copies

When a tree is copied between two parsers that are the same, for example in ESQL:

```
SET OutputRoot.XMLNSC.myTesData.Data = InputRoot.XMLNSC.myInputData.myDarta
```

Then a "*Like Parser Copy*" is performed. In this instance, the integration node can be certain that the tree can be fully represented by the target parser and so all parser information is copied. The target parser has an identical copy of the tree that is taken from the source parser. All the parser structure under this element is retained and all attributes in the message are still represented as attributes in the target tree.

Unlike Parser Copies

When you copy a tree between two different parsers, for example in ESQL:

```
SET OutputRoot.DFDL.Data.Account = InputRoot.XMLNSC.Data.Account
```

In this instance, the integration node cannot know whether the source tree can be fully represented by the target parser. This behavior is because parsers do not all support the same type of structure and content information. In this instance instead of creating a copy of the source tree, the integration node must navigate down the logical structure, creating the elements using the target parser. Therefore, that parser structure is not maintained, and elements that were attributes in the source might not be attributes in the target tree. After an unlike parser copy, the target tree consists only of Name-Value pairs under the target parser with no other parser information.

Note that any parsers that were children of the root element of the source tree that are copied, are not preserved in the output tree. When serializing, you must ensure that trees created from unlike parser copies are constructed in such a way that a body parser can serialize the output message.

Predefined and self-defining messages

Both predefined and self-defining messages are supported.

Each message that flows through an integration node has a specific structure that is meaningful to the applications that send or receive that message.

You can use:

- Messages that you have modeled in the Integration Development perspective of the IBM App Connect Enterprise Toolkit; these messages are referred to as *predefined messages*. A model-driven parser requires predefined messages.
- Messages that can be parsed without a model; these are called *self-defining messages*.

Predefined messages

When you create a message in the IBM App Connect Enterprise Toolkit, you define the fields (*Elements*) in the message, along with special field types that you might need, and specific values (Value Constraints) to which the fields might be restricted.

When you deploy a message set to an integration node, the message model is sent to the integration node in a form appropriate to the parser that is used to parse and write the message.

For information about the benefits of predefining messages, see "Why model messages?" on page 2138

Self-defining messages

You can create and route messages that are self-defining. The best example of a self-defining message is an XML document.

You can also model self-defining messages in the IBM App Connect Enterprise Toolkit. However, you do not need to deploy these message sets to the integration nodes that support those message flows. For further information about why you might want to model these messages, see "Why model messages?" on page 2138.

Which body parser should you use?

The characteristics of the messages that your applications exchange indicate which body parser you must use.

WebSphere IBM Integration provides a range of message parsers. Each parser processes either message body data for messages in a particular message domain (for example, XML), or particular message headers (for example, the MQMD).

Review the messages that your applications send to the integration node, and determine to which message domain the message body data belongs, using the following criteria as a guide.

If your application data uses SOAP-based web services, including SOAP with Attachments (MIME) or MTOM
Use the SOAP domain. The SOAP domain has built-in support for WS-Addressing and WS-Security standards.

If your application data uses JSON format, as maybe used in RESTful web services
Use the JSON domain.

If your application data is in XML format other than SOAP
The domain that you use depends on the nature of the XML documents and the processing that you want to perform. See "Which XML parser should you use?" on page 525

If your application data comes from a C or COBOL application, or consists of fixed-format binary data
Use the DFDL domain. You can also use the MRM domain with a Custom Wire Format (CWF) physical format.

If your application data consists of formatted text, perhaps with field content that is identified by tags, or separated by specific delimiters, or both
Use the DFDL domain. You can also use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is created using the JMS API
The domain that you use depends on the type of the JMS message. For a full description of JMS message processing, see "JMS message as input" on page 860.

If your application data is from a WebSphere Adapter such as the adapters for SAP, PeopleSoft, or Siebel
Use the DataObject domain.

If your application data is in SAP text IDoc format, such as those exported using the IBM MQ Link for R3

Use the DFDL domain. You can also use the MRM domain with a Tagged/Delimited String (TDS) physical format.

If your application data is in MIME format other than SOAP with Attachments (for example, RosettaNet)

Use the MIME domain. If the message is multipart MIME, you might have to parse specific parts of the message with other parsers. For example, you might use the XMLNSC parser to parse the XML content of a RosettaNet message.

If you do not know, or do not have to know, the content of your application data

Use the BLOB domain.

Which XML parser should you use?

If your messages are general-purpose XML documents, you can use one of the dedicated XML domains (XMLNSC or XMLNS) to parse the message, or you can use the MRM domain to parse the message.

Note: Although SOAP XML can be parsed using any namespace-aware XML parser, use the dedicated SOAP domain to parse SOAP XML because the SOAP domain provides full support for SOAP with Attachments, and standards such as WS-Addressing and WS-Security.

Note: The XML domain is deprecated. Do not use it for developing new message flows. The XML domain still works with existing message flows.

Which XML parser you choose depends on the nature of your XML messages, and the transformation logic that you want to use. The differentiating features of each domain are:

- The XMLNSC parser has a new architecture that gives significant performance improvements over the XMLNS and XML parsers.
- The XMLNSC parser can be used with or without an XML Schema that is generated from a message set. Using a message set with the XMLNSC parser allows the parser to operate in validating mode which provides the following capabilities:
  - XML Schema 1.0 compliant validation when parsing and writing.
  - The XML Schema indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The MRM parser must be used with a message dictionary that is generated from a message set. This message dictionary enables the MRM parser to provide the following capabilities: For example:
  - Validation against the dictionary when parsing and writing. Note that validation is not XML Schema 1.0 compliant.
  - The dictionary indicates the real data type of a field in the message instead of always treating the field as a character string.
  - Base64 binary data can be automatically decoded.
- The XMLNS parser is programmatic and does not use a model when parsing.

  This means that:
  - All data in an XML message is treated as character strings.
  - Validation is not possible when parsing and writing.
- The MRM parser uses information from the XML physical format of a message set to simplify the task of creating transformation logic:
  - Date and time information can be extracted from a data value using a specified format string.
  - When creating output messages, the MRM parser can automatically generate the XML declaration, and other XML constraints.
- The XMLNSC and XMLNS parsers do not use XML physical format information from a message set. Transformation logic must explicitly create all constructs in an output message.

- The MRM parser discards some parts of an XML message when parsing; for example, white space formatting, XML comments, XML processing instructions, and inline DTDs. If you use this parser, you cannot create these constructs when building an output message.
- The XMLNSC parser, by default, discards white space formatting, XML comments, XML processing instructions, and inline DTDs. However, options are provided to preserve all of these constructs, except inline DTDs. You can create them all, except inline DTDs, when constructing an output message.
- The XMLNS parser preserves all parts of an XML document, including white space formatting. You can create all XML constructs when constructing an output message.
- The XMLNSC and MRM parsers build compact message trees that use fewer syntax elements than the XMLNS parser for attributes and simple elements. This makes these parsers more suitable than the XMLNS parser for parsing large XML messages.
- The XMLNS parser builds a message tree that conforms more closely than the XMLNSC or MRM parsers to the XML Data Model. You might want to use this parser if you are using certain XPath expressions to access the message tree, and the relative position of parent and child nodes is important, or if you are accessing text nodes directly.
- The XMLNS parser holds the text content of an element as a child of the element, whereas the XMLNSC parser builds a compact tree. To update a text value in the compact tree of the XMLNSC parser, use ESQL to set the VALUE of the target node. For example:

```
Set OutputRoot.XMLNSC.TestMsg.Value3 VALUE = InputRoot.XMLNSC.TestMsg.Value2;
```

Tip: If performance is critical, use the XMLNSC domain.

Tip: If you need to validate the content and values in XML messages, use the XMLNSC domain.

Tip: If you need to preserve formatting in XML messages on output, use the XMLNSC domain with the option to retain mixed content.

Tip: If you require the message tree to conform as closely as possible to the XML data model, perhaps because you are using certain XPath expressions to access the message tree, use the XMLNS domain.

Tip: If you are taking non-XML data that has been parsed by the DFDL domain, and merely transforming the data to the equivalent XML, use the XMLNSC domain.

Tip: If you are taking non-XML data that has been parsed by the CWF or TDS formats of the MRM domain, and merely transforming the data to the equivalent XML, use the MRM domain. You can achieve this by adding an XML physical format to the message set with default values, and changing the Message Format in the Properties folder of the message tree.

SOAP parser and domain

You can use the SOAP parser to create a common WSDL-based logical tree format for working with web services, independent of the physical bitstream format.

Use the SOAP parser in conjunction with the SOAP nodes in your message flow.

Messages in the SOAP domain are processed by the SOAP parser. The SOAP parser creates a common logical tree representation for all SOAP-based web services and validates the message against a WSDL definition. If a runtime message is not allowed by the WSDL, an exception is thrown; otherwise, the `portType` and `operation` names from the WSDL file are saved in the logical tree.

The SOAP domain offers `WS-*` processing, together with a canonical tree shape that is independent of the wire format (XML or MIME).

The following standards are supported:

- WSDL 1.1
- SOAP 1.1 and 1.2
- MIME 1.0
- Message Transmission Optimization Mechanism (MTOM) 1.0

A WSDL 1.1 definition must be deployed to describe the web service messages that the SOAP domain needs to parse and write at run time. Therefore, the SOAP parser is always model-driven. The bitstream format for these runtime messages can be SOAP 1.1 or SOAP 1.2, optionally wrapped by MIME as an SwA (SOAP with Attachments) or MTOM message.

When an application, library, or message set that supports the SOAP domain is added to a BAR file, XML schemas are created automatically. WSDL files in the application, library, or message set are added to the BAR file. The WSDL and XML schemas are deployed to the integration node and used by the SOAP parser.

If you want the SOAP domain to parse your SOAP web service, complete the steps in the following topic: "Using the SOAP domain to parse a SOAP web service" on page 527.

Tip: The SOAP parser invokes the XMLNSC parser to parse and validate the XML content of the SOAP web service. See "XMLNSC parser" on page 531.

SOAP message details
A SOAP message consists of an <Envelope>, which is the root element in every SOAP message, and this contains two child elements, an optional <Header> and a mandatory <Body>.

If the SOAP message has attachments, the 'envelope' is wrapped by MIME, or is encoded as MTOM.

For further information on the structure of a SOAP message, see "The structure of a SOAP message" on page 807.

Using the SOAP domain to parse a SOAP web service
Use the SOAP parser to create a common WSDL-based logical tree format for working with web services.

Before you begin

Read the concept information in "SOAP parser and domain" on page 526.

About this task

To use the SOAP domain to parse your SOAP web service, complete the following steps:

Procedure

1. Create an application or library. Alternatively, you can create a message set, or locate an existing message set.

   If you are likely to reuse your imported WSDL for multiple solutions, use a shared library.
2. If you are using a message set, indicate that the message set supports the SOAP domain by either setting the default message domain project to SOAP, or select the SOAP check box (under Supported message domains).
3. To create a message root (in an application or library) or a message definition file (in a message set), import your WSDL file into the application, library, or message set.

   Message roots or message definition files for the SOAP envelope and the SOAP logical tree are also added to the application, library, or message set automatically.
4. Add the application, library, or message set to a BAR file.

   If the WSDL file is in an application or library, the WSDL files and XSD files appear directly inside the `.appzip`, `.shlibzip`, or `.libzip` file inside the BAR file. If the WSDL file is in a message set, the required XML schema and WSDL files are generated in a file with the extension `.xsdzip`.
5. Deploy the BAR file.

Results

If you associate your WSDL with a SOAP node in your message flow, the `Message domain` property on the node is automatically set to SOAP and cannot be changed. If you are using a message set, the `Message model` property is automatically set to the name of the message set that contains the WSDL and cannot be changed. If you are using an application or library, the `Message model` property is empty and cannot be changed.

You can use a WSDL file in a shared library to configure a message flow in another container (an application, for example). However, that container must reference the shared library that contains the WSDL file. If you use a WSDL file from a shared library to configure a SOAP node, the `Message model` property contains the name of the shared library in braces, {}. The `WSDL file name` property also contains the name of the shared library.

XML parsers and domains

You can use XML domains to parse and write messages that conform to the W3C XML standard.

The term *XML domains* refers to a group of three domains that are used by IBM App Connect Enterprise to parse XML documents.

When reading an XML message, the parser that is associated with the domain builds a message tree from the input bit stream. The input bit stream must be a well-formed XML document that conforms to the W3C XML Specification (version 1.0 or 1.1).

When writing a message, the parser creates an XML bit stream from a message tree.

The domains have different characteristics, for guidance about which domain to choose, see "Which XML parser should you use?" on page 525.

XMLNSC domain

The XMLNSC domain is the preferred domain for parsing all general purpose XML messages, including those messages that use XML namespaces. This parser is the preferred parser for the following reasons:

- The XMLNSC parser has an architecture that results in ultra-high performance when parsing all kinds of XML.
- The XMLNSC parser reduces the amount of memory that is used by the logical message tree that is created from the parsed message. The default behavior of the parser is to discard non-significant white space and mixed content, comments, processing instructions, and embedded DTDs; however controls are provided to retain mixed content, comments, and processing instructions, if required.
- The XMLNSC parser can operate as a model-driven parser, and can validate XML messages against XML Schemas generated from a message set, to ensure that your XML messages are correct.

XMLNS domain

If the XMLNSC domain does not meet your requirements, use the alternative namespace-aware domain and parser.

XML domain

The XML domain is not namespace-aware. It is deprecated and must not be used to develop new message flows.

The MRM domain also provides XML parsing and writing facilities. For guidance on when you might use MRM XML instead of one of the XML parsers, see "Which XML parser should you use?" on page 525.

By default, the three XML parsers are *programmatic* parsers and do not use a message set at run time when parsing and writing. However, the XMLNSC parser can operate as a model-driven parser and can validate XML messages for correctness against XML Schemas generated from a message set.

When you use the XMLNS or XML parsers, or the XMLNSC parser without a message set, it is good practice to create and use a message set in the IBM App Connect Enterprise Toolkit; this action simplifies the development of your message flow applications, even though the message set is not deployed to the integration node run time.

For the advantages of creating a message set, see "Why model messages?" on page 2138.

The XML parsers are on-demand parsers. For more information, see Parsing on demand.

The topics in this product documentation provide a summary of XML terminology, concepts, and message constructs. These aspects are important when you use XML messages in your message flows.

Tip: For more detailed information about XML, see the World Wide Web Consortium (W3C) Web site.

Example XML message parsing

A simple XML message might take the following form:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Envelope
PUBLIC "http://www.ibm.com/dtds" "example.dtd"
[<!ENTITY Example_ID "ST_TimeoutNodes Timeout Request Input Test Message">]
>
<Envelope version="1.0">
 <Header>
  <Example>&Example_ID;</Example>
  <!-- This is a comment  -->
 </Header>
 <Body  version="1.0">
  <Element01>Value01</Element01>
  <Element02/>
  <Element03>
   <Repeated>ValueA</Repeated>
   <Repeated>ValueB</Repeated>
  </Element03>
  <Element04><P>This is <B>bold</B> text</P></Element04>
 </Body>
</Envelope>
```

The following sections show the output that is created by the Trace node when this example message has been parsed in the XMLNS and XMLNSC parsers. They demonstrate the differences in the internal structures that are used to represent the data as it is processed by the integration node.

Example XML Message parsed in the XMLNS domain

In the following example, the white space elements within the tree are present because of the space, tab, and line breaks that format the original XML document; for clarity, the actual characters in the trace have been replaced with `'WhiteSpace'`. White space within an XML element does have business meaning, and is represented by using the Content syntax element. The XmlDecl, DTD, and comments, are represented in the XML domain using explicit syntax elements with specific field types.

```
(0x01000010):XMLNS        = (
    (0x05000018):XML      = (
      (0x06000011): = '1.0'
      (0x06000012): = 'UTF-8'
      (0x06000014): = 'no'
    )
    (0x06000002):         = 'WhiteSpace'
    (0x05000020):Envelope = (
      (0x06000004): = 'http://www.ibm.com/dtds'
      (0x06000008): = 'example.dtd'
      (0x05000021): = (
        (0x05000011):Example_ID = (
          (0x06000041): = 'ST_TimeoutNodes Timeout Request Input Test Message'
        )
      )
    )
    (0x06000002):         = 'WhiteSpace'
    (0x01000000):Envelope = (
      (0x03000000):version = '1.0'
      (0x02000000):         = 'WhiteSpace'
      (0x01000000):Header  = (
        (0x02000000):        = 'WhiteSpace'
        (0x01000000):Example = (
          (0x06000020): = 'Example_ID'
          (0x02000000): = 'ST_TimeoutNodes Timeout Request Input Test Message'
          (0x06000021): = 'Example_ID'
        )
        (0x02000000):        = 'WhiteSpace'
        (0x06000018):        = ' This is a comment  '
        (0x02000000):        = 'WhiteSpace'
      )
      (0x02000000):        = 'WhiteSpace'
      (0x01000000):Body    = (
        (0x03000000):version   = '1.0'
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element01 = (
          (0x02000000): = 'Value01'
        )
        (0x02000000):          = 'WhiteSpace'
```

```
        (0x01000000):Element02 =
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element03 = (
          (0x02000000):         = 'WhiteSpace'
          (0x01000000):Repeated = (
            (0x02000000): = 'ValueA'
          )
          (0x02000000):         = 'WhiteSpace'
          (0x01000000):Repeated = (
            (0x02000000): = 'ValueB'
          )
          (0x02000000):         = 'WhiteSpace'
        )
        (0x02000000):          = 'WhiteSpace'
        (0x01000000):Element04 = (
          (0x01000000):P = (
            (0x02000000):  = 'This is '
            (0x01000000):B = (
              (0x02000000): = 'bold'
            )
            (0x02000000):  = ' text'
          )
        )
        (0x02000000):          = 'WhiteSpace'
      )
      (0x02000000):        = 'WhiteSpace'
    )
```

Example XML Message parsed in the XMLNSC domain

The following trace shows the elements that are created to represent the same XML structure within the compact XMLNSC parser in its default mode. In this mode, the compact parser does not retain comments, processing instructions, or mixed text.

The example illustrates the significant saving in the number of syntax elements that are used to represent the same business content of the example XML message when using the compact parser.

By not retaining mixed text, all of the white space elements that have no business data content are no longer taking any space in the integration node message tree at run time. However, the mixed text in `Element04.P` is also discarded, and only the value of the child folder, `Element04.P.B`, is held in the tree; the text `This is` and `text` in P is discarded. This type of XML structure is not typically associated with business data formats; therefore, use of the compact XMLNSC parser is typically desirable. However, if you want to this type of processing, either do not use the XMLNSC parser, or use it with *Retain mixed text mode* enabled.

The handling of the XML declaration is also different in the XMLNSC parser. The version, encoding, and stand-alone attributes are held as child entities of the XmlDeclaration, rather than as elements with a particular field type.

```
(0x01000000):XMLNSC     = (
    (0x01000400):XmlDeclaration = (
      (0x03000100):Version    = '1.0'
      (0x03000100):Encoding   = 'UTF-8'
      (0x03000100):StandAlone = 'no'
    )
    (0x01000000):Envelope       = (
      (0x03000100):version = '1.0'
      (0x01000000):Header  = (
        (0x03000000):Example = 'ST_TimeoutNodes Timeout Request Input Test Message'
      )
      (0x01000000):Body    = (
        (0x03000100):version   = '1.0'
        (0x03000000):Element01 = 'Value01'
        (0x01000000):Element02 =
        (0x01000000):Element03 = (
          (0x03000000):Repeated = 'ValueA'
          (0x03000000):Repeated = 'ValueB'
        )
        (0x01000000):Element04 = (
          (0x01000000):P = (
            (0x03000000):B = 'bold'
          )
        )
   )
```

Some predefined message models are supplied with the IBM App Connect Enterprise Toolkit and can be imported by using the New Message Definition File wizard and selecting the IBM supplied message option. See Message Sets: IBM supplied messages that you can import.

XMLNSC parser
The XMLNSC parser is a flexible, general-purpose XML parser that offers high performance XML parsing and optional XML Schema validation.

The XMLNSC parser has a range of options that make it suitable for most XML processing requirements. Some of these options are only available in the XMLNSC parser.

Although the XMLNSC parser is capable of parsing XML documents without an XML Schema, extra features of the parser become available when it operates in model-driven mode. In model-driven mode, the XMLNSC parser is guided by an XML Schema, which describes the shape of the message tree (the logical model).

XML Schemas are created automatically from the content of a message set when the message set is added to a BAR file. The XML Schemas are deployed to the integration node and used by the XMLNSC parser to validate your XML messages. Validation is fully compliant with the XML Schema 1.0 specification.

For guidance on when to use the XMLNSC domain and parser, see "Which XML parser should you use?" on page 525.

If you want the XMLNSC domain to parse a message, select *Message Domain* as XMLNSC on the appropriate node in the message flow. Additionally, if you want the XMLNSC parser to validate your messages, perform the additional steps that are described in "XMLNSC validation" on page 537.

Features of the XMLNSC parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "XML parsers namespace support" on page 543.
On-demand parsing	Yes	See Parsing on demand.
Compact message tree	Yes	Less memory is used when building a message tree from an XML document. See "Manipulating messages in the XMLNSC domain" on page 1706.
Opaque parsing	Yes	One or more elements can be parsed opaquely. See "XMLNSC opaque parsing" on page 536.
Ultra high performance	Yes	The architecture of the XMLNSC parser means that the parser's use of processor resources is significantly less than that of the other XML parsers.
Validation	Yes	See the table that follows this one.
Inline DTD support	Partial	Inline DTDs are processed but discarded. See "XMLNSC DTD support" on page 540.

Feature	Present	Description
XML Data Model compliance	Partial	The compact nature of the message tree means that some XPath queries are not supported.

The following features are only available when message validation is enabled. See "XMLNSC validation" on page 537.

Feature	Description
Message validation	Validates compliance with the XML Schema 1.0 specification.
xsi:nil support	Sets the value of an element to NULL if it has xsi:nil="true" and the XML Schema indicates that it is nillable.
Default value support	Sets the value of an empty element, or missing attribute, to its default value, according to XML Schema rules.
Use correct simple types	Allows the use of the simple types that are defined in the XML Schema when building the message tree.
Base64 support	Converts base64 data to BLOB when parsing. Converts BLOB to base64 when writing.

If you specify the SOAP domain as the owner of a SOAP web services message, the SOAP parser invokes the XMLNSC parser in model-driven mode to parse the XML content of the SOAP message.

If you specify the DataObject domain as the owner of a WebSphere Adapter message, and the message is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

XMLNSC empty elements and null values
Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of these values by the XMLNSC domain. For details on processing null values in Graphical Data Maps and ESQL, see "Handling null values" on page 566.

Parsing

Description	XML input parsed by XMLNSC	Value of 'element' in message tree
Empty element value	`<element/>`	Empty string
Empty element value	`<element></element>`	Empty string
Folder with child elements	`<element><childElement/ ></element>`	No value
Nil element value	`<element xsi:nil="true"/>`	No value and a child `xsi:nil` attribute with the value 'true'.

Both forms of an empty element result in the same value in the message tree.

Writing

Description	Value of 'element' in message tree	XML output from XMLNSC parser
Empty element value	Empty string	`<element/>`
Null element value	NULL	`<element/>`
Folder with child elements	No value	`<element><childElement/ ></element>`
Nil element value	Empty string, NULL, or Folder	`<element xsi:nil="true"/>` Note that the XMLNSC parser produces only `xsi:nil` attributes that are already in the message tree. It does not automatically produce `xsi:nil` attributes for all message tree elements that have a NULL value and are 'nillable'.

Empty elements

An empty element can take two forms in an XML document:

```
- <element/>
- <element></element>
```

The XMLNSC parser treats both forms in the same way. The element is added to the message tree with a value of "" (the empty string).

When a message tree is produced by the XMLNSC parser, it always uses the first form for elements that have a value of "" (the empty string) in the input XML message.

Elements with an `xsi:nil` attribute

If validation is enabled for the flow, the XMLNSC parser performs the following validations:

- The 'nillable' property of an element definition in the XML schema is set to 'true'.

  If the element in the document has an `xsi:nil` attribute with the value 'true', the element must not have a value, or contain any child elements.
- The 'nillable' property of an element definition in the XML schema is set to 'false'.

  The element in the input document must not have an `xsi:nil` attribute.

If an element in the input document has an `xsi:nil` attribute, the XMLNSC parser creates an element in the message tree with no value, and a child `xsi:nil` attribute with the value 'true' or 'false'.

When a message tree is written to a bit stream by the XMLNSC parser, if the value of the element is empty, NULL, or no value, and the element has no child elements, the element is written as `<element/>`. If the element has an `xsi:nil` attribute, it is written exactly like any other attribute.

Note that the XMLNSC parser produces only `xsi:nil` attributes that are already in the message tree. It does not automatically produce `xsi:nil` attributes for all message tree elements that have a NULL value and are 'nillable'.

XMLNSC: Using field types
The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the

output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See "XMLNSC: XML Schema support" on page 1715 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060

XMLNSC Field Type constant	Purpose	Value
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.BitStream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see "Working with large XML messages" on page 1703.	0x03000200
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

XMLNSC opaque parsing

Opaque parsing is a performance feature that is offered by the XMLNSC domain.

If you are designing a message flow and you know that certain elements in a message are never referenced by the message flow, you can specify that these elements are parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

Use the property *Opaque Elements* on the *Parser options* page of the relevant message flow node to specify the elements that you want to be parsed opaquely. This property specifies a list of element names. If an element in the input XML message is named in this list, the element is parsed as a single string.

An opaque element cannot be queried like an ordinary element; its value is the segment of the XML bit stream that belongs to the element, and it has no child elements in the message tree, even though it can represent a large subtree in the XML document.

When an opaque element is serialized, the value of the element is copied directly into the output bit stream. The string is converted to the correct code page, but no other changes are made. Because this might produce a bit stream that is not valid XML, some care is required.

Do not parse an element opaquely in any of the following cases:

- The message flow must access one of its child elements.
- The message flow changes the namespace prefix in a way that affects the opaque element or one of its child elements **and** the element is to be copied to the output bit stream.
- The element, or any child element, contains a reference to an internal entity that is defined in an inline DTD **and** the element is to be copied to the output bit stream.
- The element contains child attributes that have default values that are defined in an inline DTD **and** the element is to be copied to the output bit stream.

Make sure that you check the above points before you specify an element for opaque parsing.

Using opaque parsing has some drawbacks. When opaque parsing is enabled, some parts of the message are never parsed, and XML that is either badly formed or not valid is allowed to pass through the message flow without being detected. For this reason, if you enable validation, you cannot use opaque parsing.

The XMLNS domain offers a more limited opaque parsing facility, but this is provided only to support existing applications. Use the XMLNSC domain in new message flows if you want to use opaque parsing.

Specifying opaque elements for the XMLNSC parser
Specify an element as an opaque element so that its content is ignored by the XMLNSC parser.

Before you begin

About this task

To specify the elements that are to be skipped by the XMLNSC parser:

Procedure

1. Right-click the selected message flow node, click **Properties** and select **Parser Options**.
2. At the bottom of the `XMLNSC Parser Options` panel is an area that lists the elements that have already been selected as opaque elements. Click **Add** to add an element to this list.

   A new pane **Add Opaque elements Entry** opens.
3. In the **Add Opaque elements Entry** pane, specify the new XML element that you want to be opaquely parsed.

   Each opaque element must be specified as an ESQL element name or an XPath expression of the form `//prefix:name` (or `//name`, if your input document does not contain namespaces).

   Note: A prefix is used rather than a full URI to identify the namespace; for further information, see "XPath namespace support" on page 1771.

What to do next

Click **Edit** or **Delete** to edit the list of opaque elements.

XMLNSC validation
The XMLNSC parser offers high-performance, standards-compliant XML Schema validation at any point in a message flow.

Validation of the input XML message or the message tree is performed against the XML Schemas that are deployed.

Validation is not the same as *parsing*. When parsing, the XMLNSC parser always checks that the input document is well-formed XML, according to the XML specification. If validation is enabled, the XMLNSC parser also checks that the XML document obeys the rules in the XML Schema.

Enabling XML Schema validation in a message flow

You must complete the following tasks to construct a message flow that validates an XML document in accordance with an XML Schema:

- Enable validation at the appropriate point in the message flow. This is typically achieved by setting the *Validate* property of the appropriate node to `Content and Value`. See "Validating messages" on page 610.
- Ensure that all required XML Schema files are deployed. See "Deploying XML Schemas" on page 538 later in this section.
- If the XML Schema files are deployed in a message set, you must also supply the name of the message set. Typically, you specify the message set by selecting the `Message model` property on a message flow node. If the parser is being invoked from a programming language such as ESQL or Java, the message set must be specified in the parameters of the function call.

Deploying XML Schemas

XML Schemas can be deployed within an application, in a library that is referenced by an application, or in a message set.

If the schemas are deployed as part of the application, they must be included in one of the projects that are referenced by the application.

If the schemas are deployed in a library, the application must reference that library, even if it will be deployed as part of a different application.

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134.

If the schemas are deployed in a message set, you must complete the following steps.

All XML Schemas that are used by IBM App Connect Enterprise must be created as message definition files within a message set.

To create and deploy a message set for XML Schema validation:

1. Create a new message set or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to XMLNSC, or that the XMLNSC check box under *Supported message domains* is selected, to indicate that the message set supports the XMLNSC domain.
3. Create a message definition file in the message set to represent your message. If you have an existing XML Schema or DTD that describes your message, you can import it. You can repeat this step for each message that you want to validate.
4. Add the message set to a BAR file, which generates the required XML Schema in a file with extension .xsdzip, and deploy the BAR file to the integration node.

Standards compliant validation

XMLNSC validation complies fully with XML Schema v1.0 as defined in the specifications that are available at http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/, with the following minor exceptions:

- Any floating point value that is smaller than 10E-43 is treated as zero.
- Any member of a group or complex type, that has both minOccurs > 1024 and maxOccurs > 1024, is validated as if minOccurs = 0 and maxOccurs is unbounded.

Validating XML v1.1 documents

You can validate documents that conform to the XML v1.1 specification, but support is limited by the fact that the XML Schema v1.0 documents must conform to XML v1.0.

As an example, you cannot always declare an XML v1.1 tag name in XML Schema v1.0. This limitation is not imposed by the XMLNSC parser implementation; it is a limitation of XML Schema v1.0.

Interpreting validation errors

A validation error is an error that results when the XML document breaks the rules that are defined in the XML schema. The XML Schema standard specifies exactly what these rules are, and how they should be applied. Validation errors that the XMLNSC parser issues contain information that links the error to the XML Schema rule that has been violated.

All validation errors are reported in BIP5025 or BIP5026 messages. Both messages begin with text in the following form:

```
XML schema validation error '[cvc-error key: error description]'
```

Examples:

```
'cvc-minInclusive-valid: The value "2" is not valid with respect to the minInclusive facet
with value "3" for type "po:itemCountType".'
```

```
'cvc-complex-type.2.4.a: Expecting element with local name "numItems" but saw "totalValue".'
```

To find the XML Schema rule that has been violated, open the XML Schema specification and search for the error key.

Example 1: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-minInclusive-valid'. Follow the link to the XML Schema rules for the minInclusive facet.

Example 2: Open http://www.w3.org/TR/xmlschema-1/ and search for ‘cvc-complex-type'. Follow the link to the XML Schema rules for validating the content of a complex type. In this case, the error key contains extra information. The ‘2.4.a' refers to the exact sub-rule that was violated. It should not be included when searching for the rule.

If the XML Schema specification does not provide enough information, you can find more information using a search engine. The XML Schema standard is very widely used, and many online tutorials and other resources are available.

XMLNSC validation restriction

Do not use patterns that include an unbounded wildcard character followed by a negated character class to validate string elements because an error occurs and validation fails even if the input string is valid.

For example, validation of the input string `<StringWithXSDPattern>ADDDDDA</StringWithXSDPattern>` results in error `cvc-pattern-valid` if a pattern of the following format `<xs:pattern value="A.*[^D]"/>` is used in the schema. The unbounded wildcard character is `.*` and the negated character class is `[^D]`. You can replace the negated character class with a non-negated class to avoid this error: using a pattern of the following format `A.*[ABC]` in this example will result in correct validation.

XMLNSC message tree options
The XMLNSC options that are described in this section affect the parsing of an XML document by the XMLNSC parser. They have no effect on XML output.

Retain Mixed Content

Mixed content is XML text which occurs between elements.

```
<parent>
     <childElement1>Not mixed content</childElement1>
     This text is mixed content
     <childElement2>Not mixed content</childElement2>
</parent>
```

By default, the XMLNSC parser discards all mixed content. Mixed content is retained in the message tree if you select *Retain mixed content* in the Parser options page of the input node. For further information, see 'Element with mixed content' in “XMLNSC: Element values and mixed content” on page 1712 .

Retain Comments

By default, the XMLNSC parser discards all comments in the input XML. Comments are retained in the message tree if you select *Retain comments* in the Parser options page of the input node. For further information, see 'Comments' in “XMLNSC: Comments and Processing Instructions” on page 1715 .

Retain Processing Instructions

By default, the XMLNSC parser discards all processing instructions in the input XML. Processing instructions are retained in the message tree if you select *Retain processing instructions* in the Parser

options page of the input node. For further information, see 'Processing Instructions' in “XMLNSC: Comments and Processing Instructions” on page 1715 .

Build tree using XML Schema data types

By default, the XMLNSC parser uses the CHARACTER data type for all element and attribute values that the parser creates in the message tree. However, if you are using the XMLNSC parser to validate the XML document, you can select *Build tree using XML Schema data types* in the Parser options page of the input node. This causes element and attribute values to be cast to the IBM App Connect Enterprise data type that most closely matches their XML Schema simple type. The exact mapping between XML schema types and IBM App Connect Enterprise types can be found in “XMLNSC data types” on page 540.

XMLNSC data types
Mapping between XML Schema simple types and the data types that the XMLNSC parser uses in the message tree when `Build tree using XML Schema types` is specified.

For mapping details, see ESQL to XML Schema data type mapping. Base64 decoding is automatically performed by the XMLNSC parser.

List types

In the message tree, a list type is represented as a parent node with an anonymous child node for each list item. This allows repeating lists to be handled without any loss of information.

If a list element repeats, the occurrences appear as siblings of one another, and each occurrence has its own set of child nodes representing its own list items.

XMLNSC DTD support
The input XML message might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNSC parser reads and uses information in the DTD while parsing, but does not add the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

The XMLNSC parser never adds the DTD to the message tree because the information that it contains has already been used during the parse. This behavior keeps the message tree compact and reduces CPU usage, and means that the XMLNSC parser does not always produce exactly the same document as it parsed. However, the business meaning of the output document is not altered.

If these restrictions are a problem, the XMLNS domain and parser provide full support for parsing and writing of the DTD. See “XMLNS DTD support” on page 543.

Writing

The XMLNSC parser can produce a DTD that contains entity definitions only. This behavior allows the XMLNSC parser to be used for writing out XML documents that use internal entities (the most common reason for using a DTD). See “Manipulating messages in the XMLNSC domain” on page 1706 for further details.

External DTDs

No support is offered for external DTDs

XMLNS parser
The XMLNS parser is a flexible, general-purpose XML parser.

The XMLNS parser is not model-driven and does not use an XML Schema when parsing XML documents.

For guidance on when to use the XMLNS domain and parser, see "Which XML parser should you use?" on page 525.

If you want the XMLNS domain to parse a particular message, you must select *Message Domain* as XMLNS on the appropriate node in the message flow.

Features of the XMLNS parser

Feature	Present	Description
Namespace support	Yes	Namespace information is used if it is present. No user configuration is required. See "Namespace support" on page 628.
On-demand parsing	Yes	See Parsing on demand.
Compact message tree	No	
Opaque parsing	Partial	Limited support from ESQL only for parsing a single element opaquely. See "XMLNS opaque parsing" on page 542.
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree. See "XMLNS DTD support" on page 543.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

XMLNS empty elements and null values
Empty elements and null values occur frequently in XML documents.

A robust message flow must be able to recognize and handle empty elements and null values. Similarly, elements in a message tree might have a NULL value, an empty value, or no value at all. This topic explains the parsing and writing of these values by the XMLNS domain. For advice on good ESQL or Java coding practices see "Handling null values" on page 566.

Parsing

Description	XML input parsed by XMLNS	Value of 'element' in message tree
Empty element value	<element/>	Empty string
Empty element value	<element></element>	Empty string
Folder with child elements	<element><childElement/></element>	No value
Nil element value	<element xsi:nil="true"/>	Empty string

Note that both forms of an empty element result in the same value in the message tree.

Note also that a NULL value is never put into the message tree by the XMLNS parser.

Writing

Description	Value of 'element' in message tree	XML output from XMLNS parser
Empty element value	Empty string	<element/>
Null element value	NULL	<element/>
Folder with child elements	No value	<element><childElement/></element>

Empty elements

An empty element can take two forms in an XML document:

```
- <element/>
- <element></element>
```

The XMLNS parser treats both forms in the same way. The element is added to the message tree with a value of "" (the empty string).

When a message tree is produced by the XMLNS parser, it always uses the first form for elements that have a value of "" (the empty string).

Elements with an xsi:nil attribute

The XMLNS parser treats the xsi:nil attribute exactly like any other attribute. When xsi:nil is encountered while parsing, it does not set the value of the parent element to NULL. If you require this behavior you should use the XMLNSC parser. When writing a message tree, if an xsi:nil attribute exists it will be produced in the same way as any other attribute.

XMLNS opaque parsing
Opaque parsing is a performance feature that is offered by the XMLNS domain.

XMLNS opaque parsing has been superseded by the opaque parsing feature of the XMLNSC domain. Do not use the XMLNS parser for opaque parsing unless your message flow requires features that are only offered by the XMLNS parser.

If you are designing a message flow, and you know that a particular element in a message is never referenced by the message flow, you can specify that that element is to be parsed opaquely. This reduces the costs of parsing and writing the message, and might improve performance in other parts of the message flow.

To specify that an XML element is to be parsed opaquely, use an **ESQL CREATE** statement with a PARSE clause to parse the XML document. Set the FORMAT qualifier of the PARSE clause to the constant, case-sensitive string 'XMLNS_OPAQUE' and set the TYPE qualifier of the PARSE clause to the name of the XML element that is to be parsed in an opaque manner.

The TYPE clause can specify the element name with no namespace (to match any namespace), or with a namespace prefix or full namespace URI (to match a specific namespace).

XMLNS opaque elements cannot be specified via the node properties.

Consider the following example:

```
DECLARE soap NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';

DECLARE BitStream BLOB ASBITSTREAM(InputRoot.XMLNS
                  ENCODING InputRoot.Properties.Encoding
                  CCSID InputRoot.Properties.CodedCharSetId);
```

```
--No Namespace
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE 'Body');

--Namespace Prefix
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE 'soap:Body');

--Namespace URI
  CREATE LASTCHILD OF OutputRoot
    DOMAIN('XMLNS')
          PARSE (BitStream
                 ENCODING InputRoot.Properties.Encoding
                 CCSID InputRoot.Properties.CodedCharSetId
                 FORMAT 'XMLNS_OPAQUE'
                 TYPE '{http://schemas.xmlsoap.org/soap/envelope/}:Body');
```

XMLNS DTD support
The input XML might contain an inline DTD.

Parsing

If the input XML document has an inline DTD, the XMLNS parser reads and uses information in the DTD while parsing, and adds the DTD information to the message tree.

Internal entity definitions in the DTD are used to automatically expand entity references that are encountered in the body of the document.

Attributes that are missing from the input document are automatically supplied with the default value specified in the DTD.

Writing

The XMLNS parser can produce any inline DTD that has been constructed in the message tree.

External DTDs

No support is offered for external DTDs

XML parsers namespace support
Namespaces in XML messages are supported by the XMLNSC and XMLNS parsers. Namespaces are not supported by the XML parser.

Parsing

The XMLNS and XMLNSC parsers can parse any well-formed XML document, whether or not the document contains namespaces. If elements or attributes have namespaces, those namespaces are applied to the elements and attributes in the message tree. Namespace prefix mappings are also carried in the message tree, and are used when serializing the message tree back to XML.

- If an element or attribute in the input XML has a namespace, the corresponding node in the message tree also has that namespace.
- If an element contains a namespace declaration (an xmlns attribute), a child element that contains its prefix and namespace URI is created in the message tree.

While the message is passing through a message flow, namespaces and namespace mappings can be modified using ESQL or any of the other transformation technologies that are offered by IBM App Connect Enterprise.

Writing

Namespaces and their prefixes are preserved in the message tree when parsing, and are used when the XMLNS and XMLNSC parsers convert a message tree to an XML bit stream.

- When serializing a message tree, the parser scans for namespace declarations on each XML element. If any are found, it uses them to select the namespace prefixes in the output document.
- If an element in the message tree has a namespace, but there is no in-scope namespace declaration for its namespace URI, a valid namespace prefix is automatically generated and used in the output XML. Auto-generated prefixes have the form NS1, NS2, and so on.

Tip: If an element in the message tree has a child element that is a ‘default namespace' declaration, every child of that element (whether an XML element or an XML attribute, at any nesting depth) must have a namespace. If this rule is not enforced, IBM App Connect Enterprise cannot generate correct XML output for the message tree.

XML parser
The XML domain is very similar to the XMLNS domain, but the XML domain has no support for XML namespaces or opaque parsing.

The XML domain is deprecated, but existing message flows that use the XML domain continue to work. Use the XMLNSC domain when developing new message flows.

The XML parser is not model-driven and does not use an XML Schema when parsing XML documents.

If you want the XML domain to parse a particular message, you must select *Message Domain* as XML on the appropriate node in the message flow.

Tip: The XMLNSC and XMLNS parsers both support XML messages that do not use namespaces, with no extra configuration.

Features of the XML parser

Feature	Present	Description
Namespace support	No	
On-demand parsing	Yes	See Parsing on demand.
Compact message tree	No	
Opaque parsing	No	
Ultra high performance	No	
Validation	No	
Inline DTD support	Yes	Inline DTDs are processed and retained in the message tree.
XML Data Model compliance	Yes	The resultant message tree conforms to the XML Data Model.

DFDL parser and domain

Data Format Description Language (DFDL) is an XML-based language used to define the structure of formatted data in a way that is independent from the data format itself.

IBM App Connect Enterprise provides support for a DFDL domain. The DFDL domain can be used to parse and write a wide variety of message formats, and is intended for general text and binary message formats,

including industry standards. It is not intended for parsing and writing XML or JSON formatted messages, which have their own message domains.

IBM App Connect Enterprise uses the DFDL parser to read and write messages in the DFDL domain. When reading a message, the DFDL parser interprets a bit stream by using grammar defined in a DFDL schema file, and generates a corresponding DFDL domain logical message tree in the integration node. When writing a message, the DFDL serializer generates a DFDL formatted bit stream from a DFDL domain logical message tree.

Because the DFDL parser is model-driven, it can perform validation of messages against the message model that is defined in the DFDL schema file. The level of validation that is performed by the DFDL parser is the same as the level that is defined by XML Schema 1.0; see "Validating messages" on page 610.

The DFDL parser is an on-demand parser. See Parsing on demand.

To process messages with the DFDL parser, select DFDL as the *Message Domain* on the relevant node in the message flow.

MRM parser and domain

You can use the MRM domain to parse and write a wide range of message formats.

The MRM domain can be used to parse and write a wide variety of message formats. It is primarily intended for non-XML message formats, but it can also parse and write XML. For guidance on when to consider using the MRM parser, instead of one of the XML parsers, to parse XML, see "Which XML parser should you use?" on page 525

The key features of the MRM domain are:

- Support for messages from applications that are written in C, COBOL, PL/I and other languages, by using the Custom Wire Format (CWF) physical format. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Support for text messages, perhaps with field content that is identified by tags, separated by specific delimiters, or both, by using the Tagged Delimited String (TDS) physical format. This includes industry standards such as CSV, HL7, SWIFT, EDIFACT, and X12.
- Support for XML messages, including those that use XML namespaces, by using the XML physical format.

IBM App Connect Enterprise uses the MRM parser to read and write messages that belong to the MRM domain. When reading a message, the MRM parser constructs a message tree from a bit stream. When writing a message, the MRM parser creates a bit stream from a message tree. The MRM parser is always model-driven, and it is guided by a message dictionary that describes the shape of the message tree (the logical model) and the physical layout of the bytes or characters in the bit stream (the physical format). A message dictionary is created automatically from the content of a message set when it is added to the BAR file. Therefore, when you create a message set for use with the MRM domain, you must define both the logical model and the appropriate physical format information.

The operation of the parser depends on the physical format that you have associated with the input or output message:

- For a binary message, the parser reads a set sequence of bytes according to information in the CWF physical format, and translates them into the fields and values in the message tree.
- For a text message, the parser uses a key piece of TDS physical format information called *Data Element Separation* to decide how to parse each portion of the message bit stream. This informs the parser whether the message uses delimiters, tags, fixed length elements, patterns, and so on. The parser then reads the data according to information in the TDS physical format, and translates it into the fields and values in the message tree.
- For an XML message, the parser reads the XML markup language (element tags and attributes), guided by information in the XML physical format, and translates them into the fields and values in the message tree.

Because the MRM parser is model-driven, it can perform validation of messages against the model that is defined in the deployed dictionary. The level of validation that is performed by the MRM parser is similar

to that defined by XML Schema 1.0, but is not fully compliant. If you use XML messages, and you want fully compliant XML Schema 1.0 validation, use the XMLNSC domain.

The MRM parser is an on-demand parser. See Parsing on demand.

If you want to use the MRM domain to parse a particular message:

1. Create a new message set with an appropriate CWF, TDS, or XML physical format; or locate an existing message set.
2. Ensure that the message set has its *Default message domain* set to `MRM`, or that the *MRM* check box under *Supported message domains* is selected to indicate that the message set supports the MRM domain.
3. Create a message definition file in the message set to represent your message, ensuring that both logical and physical format information is provided. If you have an existing C, COBOL, XML Schema, or DTD description of your message, you can import the description by using a wizard.
4. Add the message set to a BAR file which will generate a message dictionary for use by the MRM parser, and deploy the BAR file to the integration node.
5. Select `MRM` as *Message Domain* on the appropriate node in your message flow.
6. Additionally set values for the *Message model*, *Message*, and *Physical format* properties on the node. The *Message* property specifies the name of the message in the message definition file.

Some predefined message models are supplied with the IBM App Connect Enterprise Toolkit and can be imported by using the New Message Definition File From IBM supplied Message wizard. The CSV, ALE IDoc, and File IDoc models are specifically for use with the MRM domain. See Message Sets: IBM supplied messages that you can import.

IBM supplies predefined message sets for industry standard formats SWIFT, X12, EDIFACT, and FIX. For more information, contact Dublin Adapters at dubadapt@ie.ibm.com.

DataObject parser and domain

Use the DataObject domain to parse and write messages for WebSphere Adapters and CORBA applications.

- "Using the DataObject domain with WebSphere Adapters" on page 546
- "Using the DataObject domain with CORBA" on page 547

Using the DataObject domain with WebSphere Adapters

You must use the DataObject domain when you use WebSphere Adapter nodes in your message flow.

IBM App Connect Enterprise uses the DataObject parser to read and write message from Enterprise Information Systems (EIS) such as SAP, PeopleSoft, and Siebel. Such messages belong to the DataObject domain.

When it receives a message from an adapter, the DataObject parser constructs a message tree from the business object that it receives from the EIS. When it writes a message, the DataObject parser creates from the message tree the business object that it sends to the EIS. The DataObject parser is always *model-driven*, and it is guided by the XML Schemas that model the EIS business objects. The XML Schemas are created automatically from the content of a message set when the message set is added to the BAR file.

If you want to parse a message using the DataObject domain, you must:

1. Create a message set, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `DataObject`, or the *DataObject* check box (under *Supported message domains*) is selected, to indicate that the message set supports the DataObject domain.
3. Create a message definition file in the message set to represent your EIS business object. Use the *New adapter connection* wizard to connect to the EIS and retrieve the Business object metadata.

4. Add the message set to a BAR file, which generates XML Schema for the DataObject parser to use, and deploy the BAR file to the integration node.
5. If you associate your adapter inbound or outbound message with an adapter node in your message flow, the *Message model* property is automatically set in the node. The *Message domain* property is always pre-selected as `DataObject`.

Tip: If a message that belongs to the DataObject domain is written to a destination other than a WebSphere Adapter, the DataObject parser invokes the XMLNSC parser to write the message as XML.

Using the DataObject domain with CORBA

You must use the DataObject domain when you use CORBA nodes in your message flow.

IBM App Connect Enterprise uses the DataObject parser to read and write message from CORBA applications. Such messages belong to the DataObject domain.

For information about how to build the tree under the DataObject domain for use with CORBA, see "CORBA operation parameters" on page 1160 and "IDL data types" on page 1156.

JMS parsers and domains

The JMSMap and JMSStream domains can be used for modeling messages that are produced by the implementations of the Java Messaging Service standard, version 1.1 or 2.0.

Use the JMSMap domain when handling JMS messages of type MapMessage. Use the JMSStream domain when handling JMS messages of type StreamMessage.

These message types appear in the integration node in XML format, and are therefore supported in an identical way to XML domain messages.

For a full description of JMS MapMessage and StreamMessage processing, see "Using JMS in IBM App Connect Enterprise" on page 852.

MIME parser and domain

Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME (Multipurpose Internet Mail Extension) parser does not support the full MIME standard, but does support common uses of MIME. You can send the messages to the integration node over HTTP or over other transport types, such as IBM MQ. Use the MIME domain if your messages use the MIME standard for multipart messages.

The MIME domain does not support Content-Type values with a media type of *message*.

To specify that a message uses the MIME domain, select `MIME` as the *Message Domain* on the relevant message flow node.

Use the MIME domain and parser to parse and write MIME messages. The MIME parser creates a logical tree, and sets up the integration node ContentType property. You can use Compute nodes and JavaCompute nodes to manipulate the logical tree. Set the Content-Type value using the ContentType property in the MIME domain.

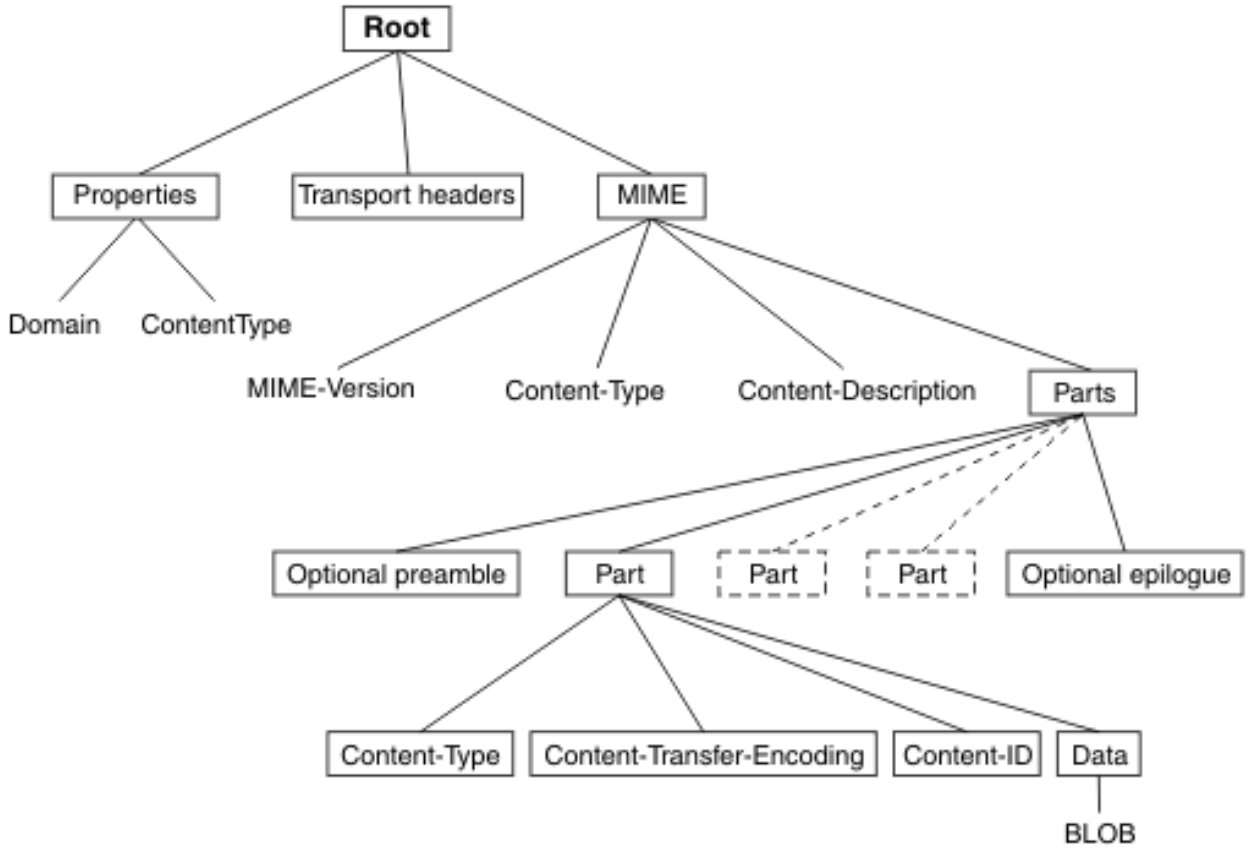
Example MIME message

The following example shows a simple multipart MIME message. The message shown is a SOAP with Attachments message with two parts: the root part and one attachment part. The boundary string *MIME_boundary* delimits the parts.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
Content-Description: Optional description of message.

Optional preamble text
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <rootpart@example.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

    <SOAP-ENV:Header xmlns:ins="http://myInsurers.com">
       <ins:ClaimReference>abc-123</ins:ClaimReference>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body xmlns:ins="http://myInsurers.com">
       <ins:SendClaim>
          <ins:ClaimDetail>myClaimDetails</ins:ClaimDetail>
          <ins:ClaimPhoto>
            <href>cid:claimphoto@example.com</href>
          </ins:ClaimPhoto>
       </ins:SendClaim>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <claimphoto@example.com>

myBinaryData
--MIME_boundary--
Optional epilogue text
```

Example MIME logical tree

The following diagram shows a MIME logical tree, which is described in “MIME tree details” on page 551. A MIME logical tree does not need to contain all of the children that are shown in the diagram. The value of the Content-Type header of a MIME message is the same as the ContentType field in the Properties subtree. The Transport-headers are headers from the transport that is used, such as an MQMD or HTTP.



You can further parse the BLOB data in the tree (for example, by using an ESQL CREATE statement) if you know about the format of that MIME part. You might be able to find information about the format from its Content-Type field in the logical tree. Alternatively, you might know the format that your MIME messages take, and be able to parse them appropriately. For example, you might know that the first MIME Part is always an XML message, and that the second MIME Part is a binary security signature.

When the EmailInput node receives an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP), the body of the email message, and any attachments, are propagated in the MIME domain. All other information relating to the email is stored in the `Root.`*`Transport headers`* MIME logical tree; for example, `Root.EmailInputHeader.To`. Where To is the storage location of one of the email elements. For a complete list of the email elements that are propagated in the MIME logical tree when you use an EmailInput node, see node.

You must specify how to parse other message formats, such as tagged delimited or binary data, within your message flow, because the MIME parser does not do this. You must also specify how to handle encoded and signed message parts, because the MIME parser does not process these.

Some pre-defined MIME message models are supplied with the IBM App Connect Enterprise Toolkit and can be imported using the New Message Definition From IBM Supplied Message wizard.

MIME messages
A MIME message consists of both data and metadata. MIME metadata consists of HTTP-style headers and MIME boundary delimiters.

MIME headers

Each header is a colon-separated name-value pair on a line. The ASCII sequence <CR><LF> terminates the line. A sequence of these headers, called a header block, is terminated by a blank line: <CR><LF><CR><LF>. Any headers that are in this HTTP style can appear in a MIME document. Some common MIME headers are described in MIME standard header fields.

Content-Type

The only header that must be present is the *Content-Type* header. This header specifies the type of the data in the message. If the Content-Type value starts with "multipart", the message is a multipart MIME message. For multipart messages the Content-Type header must also include a boundary attribute that gives the text that is used to delimit the message parts. Each MIME part has its own Content-Type field that specifies the type of the data in the part. This can also be multipart, which allows multipart messages to be nested. MIME parts with any other Content-Type values are handled as BLOB data.

If a MIME document is sent over HTTP, the Content-Type header appears in the HTTP header block rather than in the MIME message body. For this reason, the integration node manages the value of the Content-Type header as the ContentType property in the *Properties* folder of the logical tree. This allows the MIME parser to obtain the Content-Type value for a MIME document that is received over HTTP. If you need to either create a new MIME tree or modify the value of the Content-Type, set the Content-Type value using the ContentType property in the MIME domain. If you set the Content-Type value directly in the MIME tree or HTTP tree, this value might be ignored or used inconsistently. The following ESQL is an example of how to set the integration node ContentType property:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Parsing

The MIME domain does not enforce the full MIME specification. Therefore, you can work with messages that might not be valid in other applications. For example, the MIME parser does not insist on a **`MIME-Version`** header. The MIME parser imposes the following constraints:

- The MIME headers must be properly formatted:
  - Each header is a colon-separated name-value pair, on a line of its own, terminated by the ASCII sequence <CR><LF>.
  - The header line must use 7-bit ASCII.
  - Semicolons are used to separate parameters:

    ```
    Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml
    ```

  - A header might contain a comment in parentheses, for example:

    ```
    MIME-Version: 1.0 (Generated by XYZ)
    ```

- A line that starts with white space is treated as a continuation of the previous line. Therefore, a long header can be split across more than one line.
- If two or more headers in a header block have the same name, their values are concatenated into a comma-separated list.
- A top-level MIME Content-Type header must be available. The header is not case-sensitive. If the transport is HTTP, any Content-Type value in the HTTP header is used as the top-level Content-Type. If the transport is not HTTP, the Content-Type must appear in the initial header block of the MIME message.
- The Content-Type value is a media type followed by the `/` character and a subtype. Examples of this are `text/xml` and `multipart/related`. The parser does not validate subtypes. The Content-Type value can be followed by one or more parameters that are separated by semicolons.
- If the media type of a message is multipart, a boundary attribute must provide the text that is used to delimit the separate MIME parts.
- Each individual MIME part can have its own Content-Type header. The part header can have a media type of multipart, so that multipart messages can be nested. In this case, a valid boundary attribute must be provided, and its value must be different from any that has been previously defined in the message. MIME parts that have any other Content-Type value are handled as BLOB data.
- MIME multipart boundary delimiters are represented in 7-bit ASCII. The boundary delimiter consists of a line starting with a hyphen pair, followed by a boundary string. This sequence must not occur

within the MIME message at any point other than as a boundary. A MIME end-delimiter is a hyphen pair, followed by the MIME boundary string, followed by a further hyphen pair. All delimiter lines must end in the ASCII sequence <CR><LF>. An example of a delimited message is:

```
--MIME_boundary
message data
--MIME_boundary
message data
--MIME_boundary--
```

where *MIME_boundary* is the boundary delimiter string, and *message data* represents message data.

- The MIME media type *message* is not supported and results in an error at run time.
- Any preamble data (text between the initial MIME header block and the first boundary delimiter) or epilogue data (text after the final boundary delimiter) is stored in the logical tree as a value-only element. Preamble data and epilogue data can appear only as the first and last children, respectively, of a Parts node.
- The MIME parser does not support on-demand parsing and ignores the *Parse Timing* property. The parser does not validate MIME messages against a message model, and ignores the IBM App Connect Enterprise Toolkit *Validate* property.

Special cases of multipart MIME

The MIME parser is intended primarily for use with multipart MIME messages. However, the parser also handles some special cases:

- Multipart MIME with just one part. The logical tree for the MIME part saves the Content-Type and other information as usual, but the Data element for the attachment is empty.
- Single-part MIME. For single-part MIME, the logical tree has no Parts child. The last child of the MIME tree is the Data element. The Data element is the parent of the BLOB that contains the message data.
- MIME parts with no content.

Secure MIME (S/MIME)

S/MIME is a standard for sending secure email messages. S/MIME has an outer level Content-Type of *multipart/signed* with parameters **protocol** and **micalg** that define the algorithms that are used to encrypt the message. One or more MIME parts can have encoded content. These parts have Content-Type values such as *application/pkcs7-signature* and a Content-Transfer-Encoding of *base64*. The MIME domain does not attempt to interpret or verify whether the message is signed.

MIME tree details
A MIME message is represented in the integration node as a logical tree. When it writes a message, the MIME parser creates a message bit stream by using the logical message tree.

Logical tree elements

A MIME message is represented in the integration node as a logical tree with the following elements:

- The root of the tree is a node called MIME.
- All correctly formatted headers are stored in the logical tree, regardless of whether they conform to the MIME standard. The headers appear in the logical tree as name=value, as shown here:

```
Content-Type=text/xml
```

- A multipart MIME message is represented by a subtree with a root node called Parts.
- Any preamble or epilogue data associated with a multipart MIME message is represented by value-only elements appearing as the first and last children of Parts.
- In the special case of single-part MIME, the content is represented by a subtree with the root called Data.

- Each part of a multipart MIME message is represented by an element called Part with a child element for each MIME header, and a last child called Data.
- The Data element represents the content of a MIME part. This makes it easier to test for the presence of body content by using ESQL because the Data element is always the last child of its parent.

Writing MIME messages

When it writes a message, the MIME parser creates a message bit stream by using the logical message tree. The MIME domain does not enforce all of the constraints that the MIME specification requires, therefore it might generate MIME messages that do not conform to the MIME specification. The constraints that the MIME parser imposes are:

- The tree must have a root called MIME, and constituent Parts, Part, and Data elements, as described in "Logical tree elements" on page 551.
- Exactly one Content-Type header must be present at the top level of the tree, or be available by using the ContentType property. Media subtypes are not validated.
- If the media type is *multipart*, a valid boundary parameter must also exist.
- Any constituent MIME parts can have exactly one Content-Type header. If the value of this header starts with *multipart*, it must also include a valid boundary parameter. The value of this boundary parameter must not be the same as other boundary parameter values in the definition.
- The MIME Content-Type value "message" is not supported and results in an error at run time.
- All name-value elements in the tree are written as `name: value` followed by the ASCII sequence <CR><LF>.

If you have other elements in the tree, the parser behaves in the same way as the HTTP header parser:

- A name-only element or a NameValue element with a NULL value results in `Name: NULL` .
- Any children of a name-value element are ignored.

The message flow must serialize subtrees if they exist; you can use the ESQL command **ASBITSTREAM**.

BLOB parser and domain

The BLOB message domain includes all the messages with content that cannot be interpreted and subdivided into smaller sections of information.

Messages in this domain are processed by the BLOB parser. The BLOB parser is a program that interprets a bit stream or message tree that represents a message that belongs to the BLOB domain. The parser then generates the corresponding tree from the bit stream on input, or a bit stream from the tree on output.

A BLOB message is handled as a single string of bytes, and although you can manipulate it, you cannot identify specific pieces of the byte string using a field reference, in the way that you can with messages in other domains.

You can process messages in the BLOB domain in the following ways:

- You can refer to the message content if you know the location (offset) of particular information within the message. You can specify offset values in ESQL statements within nodes in a message flow to manipulate the information.
- You can store the message in an external database, in whole or in part (where the part is identified by the offset of the data that is to be stored).
- You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data. The BLOB message cannot be:
  - The message content in a message where `Content Validation` is defined as Open or Open Defined (for example, the message body of a SOAP envelope)
  - The message represented by a wildcard inside another message

  The UnknownParserName field is ignored.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, which has a child element, also called BLOB, which contains the data.

For example, `InputBody.BLOB.BLOB[10]` identifies the tenth byte of the message body; `substring(InputBody.BLOB.BLOB from 10 for 10)` references 10 bytes of the message data starting at offset 10.

If you want to use the BLOB parser to parse a particular message, select `BLOB` as the `Message Domain` on the relevant node in your message flow.

IDOC parser and domain

The IDOC domain can be used to process messages that are sent to the integration node by SAP R3 clients across the IBM MQ link for R3. Such messages are known as SAP ALE IDocs.

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format. See "MRM parser and domain" on page 545.

A typical ALE IDoc message that has been sent from SAP to the WebSphere MQ link for R3 consists of an MQMD header, an MQSAPH header, and the ALE IDoc itself. The IDoc is made up of fixed size structures:

- The first structure is the Control Structure (DC). This is a complex element 524 bytes long that contains a fixed set of SAP-defined simple elements.
- One or more Data Structures (DDs). Each DD is a complex element 1063 bytes long that contains a fixed set of SAP-defined simple elements that occupies 63 bytes, followed by 1000 bytes of user-defined segment data.

IBM App Connect Enterprise uses the IDOC parser to read and write ALE IDocs that belong to the IDOC domain. When reading a message, the IDOC parser constructs a message tree from a bit stream. When writing a message, the IDOC parser creates a bit stream from a message tree.

The IDOC parser processes the SAP-defined elements in the DC, then, for each DD, the IDOC parser processes the SAP-defined elements, then calls the MRM parser to process the user-defined segment data, using its CWF physical format. The IDOC parser is therefore a model-driven parser, and requires that you create a message model in which to model the IDoc message, and deploy it to the integration node.

If you want the IDOC domain to parse a particular message, you must:

1. Create a new message set with a CWF physical format, or locate an existing message set.
2. Ensure that either the message set has its *Default message domain* project set to `IDOC`, or the *IDOC* check box (under *Supported message domains*) is selected, to indicate that the message set supports the IDOC domain.
3. Create message definition files in the message set to represent your message. See Building the message model for the IDOC parser for the steps involved.
4. Add the message set to a BAR file which generates a message dictionary for use by the MRM parser, and deploy the BAR file to the integration node.
5. Select *Message Domain* as `IDOC` on the appropriate node in your message flow.
6. Additionally, select *Message model* and *Physical Format* on the node. (You do not need to select *Message*).

JSON parser and domain

JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language.

IBM App Connect Enterprise supports a JSON domain. Messages in the JSON domain are processed by the JSON parser and serializer. The JSON parser interprets a bit stream by using the JSON grammar, and generates a corresponding JSON domain logical message tree. When processing data for output, the JSON serializer generates a JSON formatted bit stream from a JSON domain logical message tree.

JSON messages can be validated against a JSON schema file or OpenAPI definition during parsing. JSON message trees can be validated against a JSON schema file or OpenAPI definition during writing or mid-flow.

You can use the JSON editor in the IBM App Connect Enterprise Toolkit to edit JSON files; however, the validation capability of the editor has a number of limitations and should not be used to validate your JSON documents. For information about how to validate JSON documents, see "JSON validation" on page 561.

JSON is a language-independent text format, based on two structures:

- Objects (name-value pairs) with the following types:
  - String
  - Number
  - Boolean
  - Null
- Ordered collections of values (arrays)

Objects and arrays can be nested.

The JSON parser accepts only a JSON object or a JSON array as the top-level value type in both an input bit stream and an output bit stream. It is not possible to use the JSON parser to output a JSON string literal, number, Boolean, or null value without first embedding it in an object or array.

For more information about JSON message structure, see "JSON message details" on page 558. For more information about JSON parser parameters, see Parameter values for the JSON parser.

JSON data streams can be parsed from any coded character set ID (CCSID) that is supported by the integration node. The data stream is parsed according to the CCSID that is defined by the transport when the JSON parser is invoked from an input or request node, or defined by the CCSID parameter in a PARSE clause on a CREATE function call. If no CCSID is specified, or if a value of 0 is set, the parser attempts to detect (by examining the first few bytes of the data stream) whether one of the following Unicode encodings is being used:

- UTF-8
- UTF-16BE
- UTF-16LE
- UTF-32BE
- UTF-32LE

If a UTF-* CCSID is explicitly specified, the JSON parser tolerates the corresponding Byte Order Mark (BOM) at the beginning of the data stream.

JSON data streams are parsed into a logical message tree under the `Data` element below the `JSON` parser root. The logical tree structure is shown in the "Example JSON message" on page 555.

The `Data` element can be accessed and manipulated from ESQL as `JSON.Data`, from Java as `JSON/Data`, from the Graphical Data Mapping editor by using the Add User-Defined function to define `JSON->Data`, or from XPath as `$Body/Data`. The JSON parser issues an error if a bit stream is not formatted according to the JSON grammar.

The JSON serializer serializes message trees into a JSON format data stream. The CCSID can be defined by either the integration node properties tree, or by transport headers in the message assembly. If no CCSID is defined, the serializer defaults to the queue manager default CCSID for all nodes except HTTP nodes, which default to UTF-8 encoding.

When the JSON serializer is invoked through an `ASBITSTREAM` function call, the CCSID is defined by the CCSID parameter. If no CCSID parameter is provided, or if the value is set to 0, the JSON serializer defaults to using UTF-8 encoding.

If you code the ASBITSTREAM function with the parser mode option set to `FolderBitStream` to parse a JSON domain message tree to a bit stream, the generated bit stream is a JSON format that is built from the target element and its children.

To process messages with the JSON parser, select **JSON** as the *Message Domain* on the relevant node in the message flow. The XSLTransform node does not support the JSON domain.

The integration node sets the HTTP Content-Type header to `application/json` when it serializes a JSON message tree, unless an explicit value is set by the message flow.

JSON objects are modeled in the integration node message tree as a sequence of *NameValue* elements. The parser builds the message tree in the order in which it encounters the members in the bit stream. The serializer writes the object members into the bit stream in the tree order.

JSON arrays are modeled in the integration node message tree as a *Name* element with a JSON parser-specific type flag of *JSON.Array* and ordered children. The children can be any of the following types of array:

- An array that contains simple values; for example:

  ```
  "array1" : [ "thing1", 1 ]
  ```

  The following message tree is produced:

  ```
  (0x01001000:Array): array1 = (
      (0x03000000:NameValue):Item = 'thing1' (CHARACTER)
      (0x03000000:NameValue):Item = 1 (INTEGER)
   )
  ```

  The JSON parser assigns the name `Item` to the *NameValue* elements.
- An array that contains objects; for example:

  ```
  "array2" : [ {"a" : 1}, {"b" : 2} ]
  ```

  The following message tree is produced:

  ```
  (0x01001000:Array):array2 = (
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):a = 1 (INTEGER)
    )
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):b = 2 (INTEGER)
    )
  )
  ```

- A multidimensional array; for example:

  ```
  "array3" : [ [1.1], [2.1] ]
  ```

  The following message tree is produced:

  ```
  (0x01001000:Array):array3 = (
    (0x01001000:Array):Item = (
      (0x03000000:NameValue):Item = 1.1E+0 (FLOAT)
    )
    (0x01001000:Array):Item = (
      (0x03000000:NameValue):Item = 2.1E+0 (FLOAT)
    )
  )
  ```

Example JSON message

The following example shows a simple JSON message:

```
{
    "name" : "John Doe",
    "age" : -1.0,
    "known" : false,
```

```
    "address" : { "street" : null,
                  "city" : "unknown" },
    "belongings" : ["item1", "item2", "item3"]
}
```

This JSON input produces the following integration node logical message tree:

```
(0x01000000:Object):JSON            = ( ['json' : 0xd55fc8]
  (0x01000000:Object):Data = (
    (0x03000000:NameValue):name       = 'John Doe' (CHARACTER)
    (0x03000000:NameValue):age        = -1E+0 (FLOAT)
    (0x03000000:NameValue):known      = FALSE (BOOLEAN)
    (0x01000000:Object   ):address    = (
      (0x03000000:NameValue):street = NULL
      (0x03000000:NameValue):city   = 'unknown' (CHARACTER)
    )
    (0x01001000:Array    ):belongings = (
      (0x03000000:NameValue):Item = 'item1' (CHARACTER)
      (0x03000000:NameValue):Item = 'item2' (CHARACTER)
      (0x03000000:NameValue):Item = 'item3' (CHARACTER)
    )
  )
)
```

JSONP support in the JSON domain
JSONP (JavaScript Object Notation with Padding) is an extension of the JavaScript Object Notation (JSON) format.

IBM App Connect Enterprise provides support for JSONP services. A JSONP service, or Remote JSON Service, is a web service that returns JSON data padded with a user-defined JavaScript function call. The JSONP response message can be interpreted as an executable script, so this functionality can be used to create cross-domain function calls.

For example:

```
http://brokerhost:7080/flowUrlPathSuffix?jsonp=scriptFn
```

This URL includes a query string, where:

- **jsonp** tells the JSONP service that any response from the URL must be returned as a JSONP message
- **scriptFn** is the name of a client-side executable function

Responses to the URL would therefore be in the JSONP format:

```
scriptFn(response)
```

The JSON message tree provides a top level `Padding` element, into which the JSON parser places the name of the client-side JSONP function. Similarly, the JSON serializer pads a JSON message if the top-level element `Padding` is present in the tree.

For more information about JSON, see “JSON parser and domain” on page 553.

For information about how to use IBM App Connect Enterprise to provide a JSONP service, see “Providing a JSONP service” on page 557.

For information about how to use IBM App Connect Enterprise to consume a JSONP service response, see “Consuming a JSONP service response” on page 558.

Example JSONP message

The following example shows a simple JSONP message:

```
scriptFn (
    {
        "name" : "John Doe",
        "age" : -1.0,
        "known" : false,
        "address" : { "street" : null,
                      "city" : "unknown" },
```

```
        "belongings" : ["item1", "item2", "item3"]
    }
)
```

This JSONP input produces the following integration node logical message tree:

```
  (0x01000000:Object):JSON            = ( ['json' : 0xd55fc8]
    (0x03000000:NameValue):Padding = 'scriptFn' (CHARACTER)
    (0x01000000:Object   ):Data    = (
      (0x03000000:NameValue):name       = 'John Doe' (CHARACTER)
      (0x03000000:NameValue):age        = -1E+0 (FLOAT)
      (0x03000000:NameValue):known      = FALSE (BOOLEAN)
      (0x01000000:Object   ):address    = (
        (0x03000000:NameValue):street = NULL
        (0x03000000:NameValue):city   = 'unknown' (CHARACTER)
      )
      (0x01001000:Array    ):belongings = (
        (0x03000000:NameValue):Item = 'item1' (CHARACTER)
        (0x03000000:NameValue):Item = 'item2' (CHARACTER)
        (0x03000000:NameValue):Item = 'item3' (CHARACTER)
      )
    )
  )
```

Providing a JSONP service
Configure your IBM App Connect Enterprise message flow to provide a JSONP service response.

Before you begin

Before completing this task, read the following overview topics about JSON:

- "JSON parser and domain" on page 553
- "JSONP support in the JSON domain" on page 556

About this task

You can use ESQL or Java to configure your message flow to provide a JSONP response.

The code examples in this task assume that the client application provides a JavaScript call in the following format:

```
<script type="text/javascript"
 src="http://brokerhost:7080/flowUrlPathSuffix?jsonp=scriptFn">
</script>
```

Procedure

1. On the **Advanced** tab of your HTTPInput node, select `Parse Query String`.
   This option enables you to access the JSONP script prefix that is included in the incoming URL, for example `scriptFn`, from the local environment tree.
2. Insert the following code, as appropriate:
   - If your message flow uses a Compute node:

     ```
     SET OutputRoot.JSON.Padding = InputLocalEnvironment.HTTP.Input.QueryString.jsonp;
     SET OutputRoot.JSON.Data.objectName = 'thing1';
     ```

   - If your message flow uses a JavaCompute node:

     ```
     MbMessage outMessage = new MbMessage();
     MbElement outRoot = outMessage.getRootElement();
     MbElement outParser = outRoot.createElementAsLastChild(MbJSON.PARSER_NAME);
     String paddingString =
         assembly.getLocalEnvironment().getRootElement().getFirstElementByPath("HTTP/Input/
     QueryString/jsonp").getValueAsString();
     MbElement padding = outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
      "Padding", paddingString);
     MbElement data = outParser.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
     ```

```
data.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,"objectName","thing1");
```

This code generates the following bit stream, sent as the HTTP reply:

```
scriptFn( {"objectName":"thing1"} )
```

This bit stream causes the JavaScript function `scriptFn` to be called with the JSON object as a parameter.

Consuming a JSONP service response
When a message flow is configured to use the JSON domain, the JSON parser automatically detects JSONP messages. The JSON parser puts the JSON padding in the top level `Padding` element, and the JSON data under the `Data` element.

Before you begin

Before completing this task, read the following overview topics about JSON:

- "JSON parser and domain" on page 553
- "JSONP support in the JSON domain" on page 556

About this task

You can process both JSON and JSONP messages in a single message flow, because the parser puts the JSON data under the `Data` element in the message tree. If JSONP padding is detected, the name of the client-side script is placed in the top level `Padding` element.

Follow these steps to test for the presence of padding:

Procedure

1. Create a message flow with an HTTPInput node, an HTTPReply node, and your choice of a Computeor JavaCompute node.
2. On the **Input Message Parsing** tab of your HTTPInput node, set the `Message domain` property to `JSON : For JavaScript Object Notation messages`.
3. Insert the following code, as appropriate:
   - If your message flow uses a Compute node:

     ```
     DECLARE PaddingRef REFERENCE TO InputRoot.JSON.Padding
     IF LASTMOVE(PaddingRef) THEN
        -- JSON Padding is present
     ELSE
         -- No JSON Padding present
     END IF;
     ```

   - If your message flow uses a JavaCompute node:

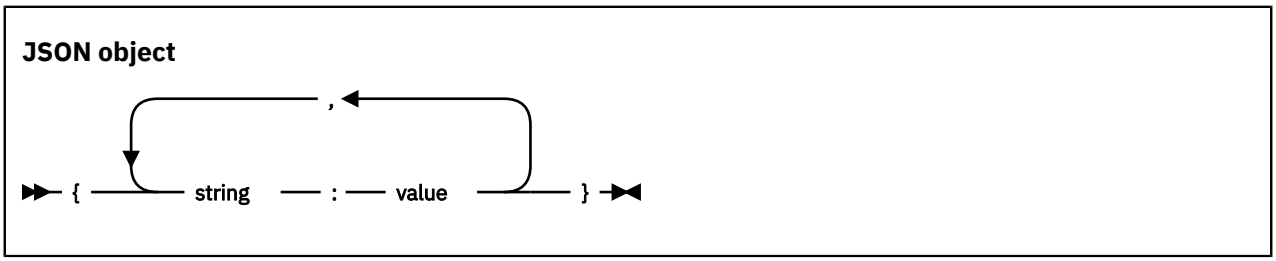
     ```
     if (message.getRootElement().getFirstElementByPath("JSON/Padding") != null){
         //JSON Padding is present
     }
     else{
         //No JSON Padding
     }
     ```

JSON message details
A JSON message consists of name-value pairs (objects), and ordered collections of values (arrays). Objects, arrays, or both structures can be nested.

For more detailed information about JSON, see the JavaScript Object Notation (JSON) web site.

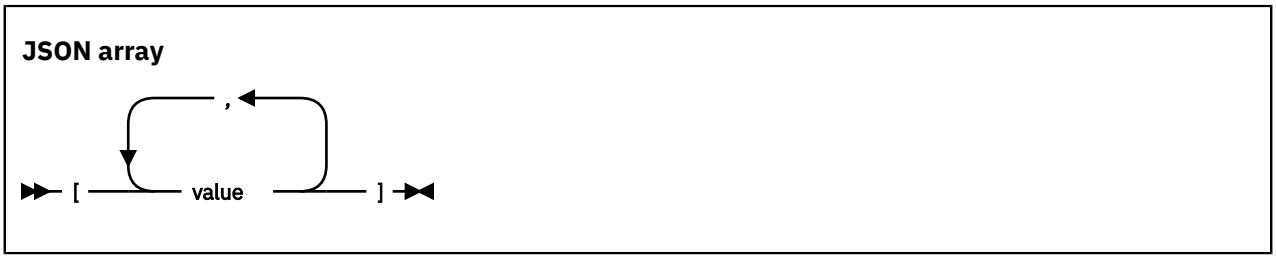
JSON object

In a JSON message, an object is an unordered set of comma-separated name-value pairs that begins with a left brace ({) and ends with a right brace (}). Each name is followed by a colon (:).



JSON array

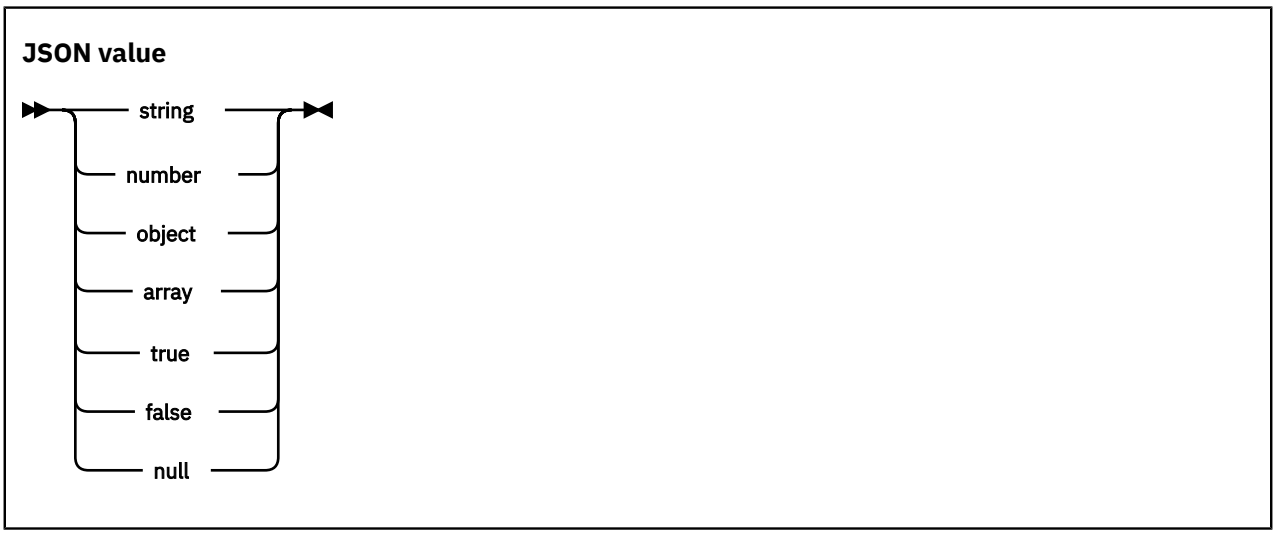
A JSON array is an ordered collection of comma-separated values that begins with left bracket ([) and ends with right bracket (]).



JSON value

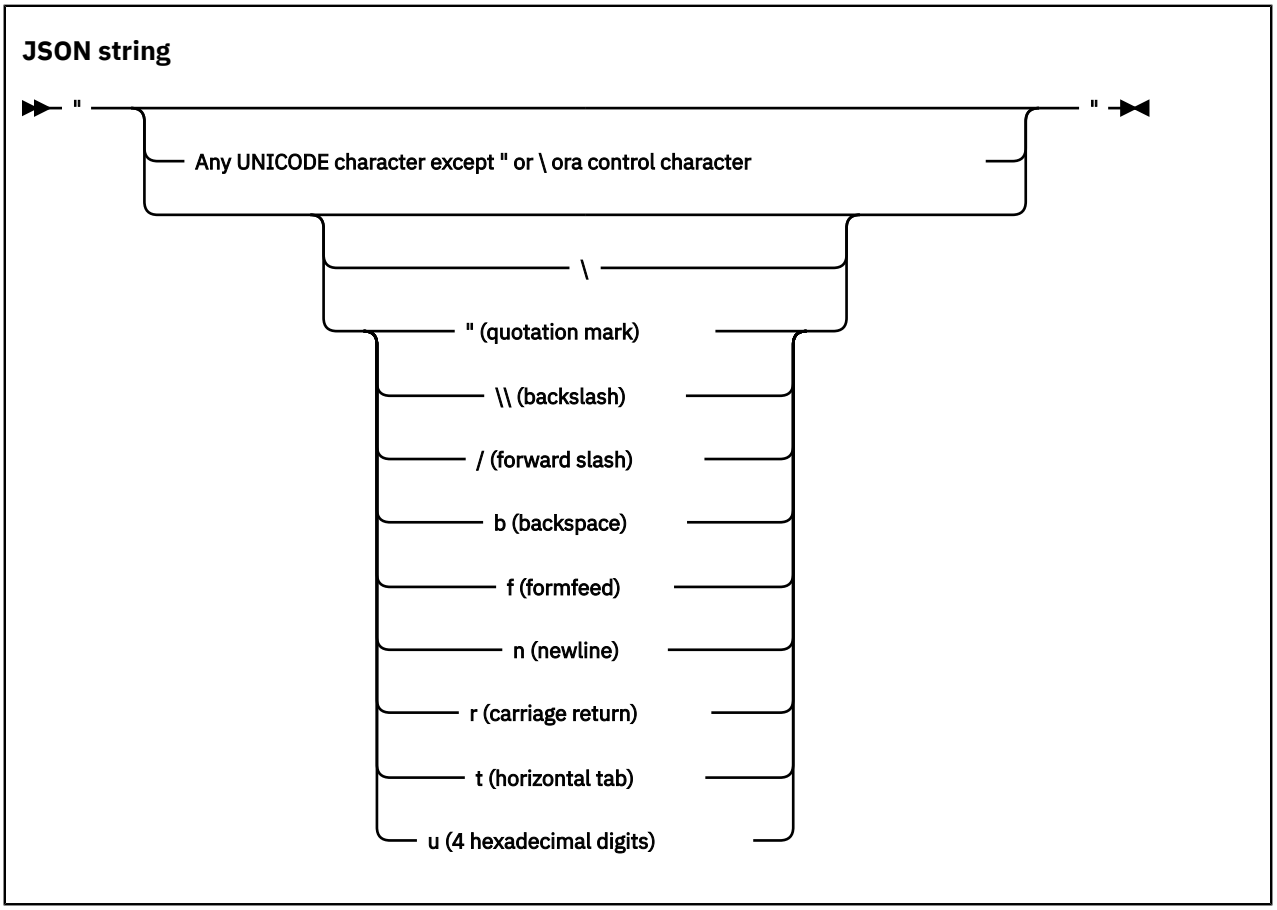
A JSON value can be any of the following structures, any of which can be nested:

- A string in double quotation marks
- A number
- Boolean
- Null
- An object
- An array



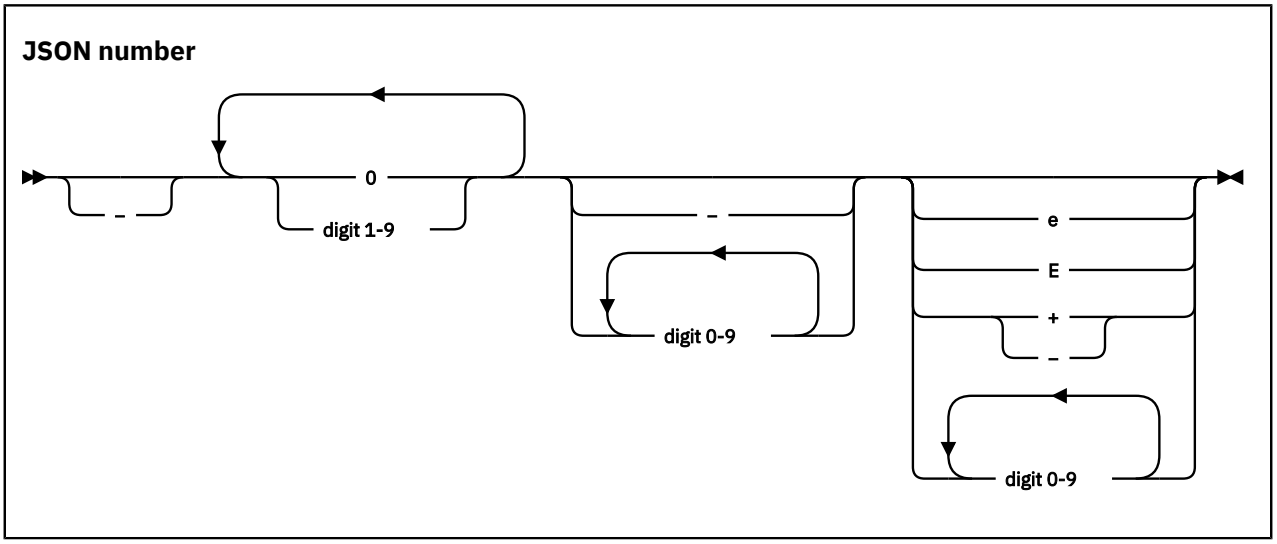
JSON string

A JSON string is very much like a C or Java string. A string is a collection of zero or more Unicode characters, wrapped in double quotation marks, using backslash escapes. A character is represented as a single character string.



JSON number

A JSON number is the same as a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens.

JSON validation
The JSON parser offers high-performance, standards-compliant JSON schema validation at any point in a message flow.

Validation of the input JSON message or the message tree is performed against the JSON schema files or OpenAPI definition files that are deployed. JSON schema must be contained in a file with a `.json` file extension, and it must either contain schema in the name (for example, `.schema.json`) or the first line of the file must indicate that it is a schema (for example, `"$schema": "http://json-schema.org/draft-04/schema#"`).

In the following sections, the term *JSON schema* is used to refer to either a JSON schema file or an OpenAPI definition file.

Validation is not the same as *parsing*. When parsing, the JSON parser always checks that the input document is well-formed JSON, according to the JSON specification. If validation is enabled, the JSON parser also checks that the JSON document obeys the rules in the JSON schema.

You can use the JSON editor in the IBM App Connect Enterprise Toolkit to edit JSON files; however, the validation capability of the editor has a number of limitations and should not be used to validate your JSON documents.

Video: JSON Validation

This video demonstrates the capability to validate a JSON message using JSON Schema.

For more videos about JSON validation, see the IBM App Connect Enterprise playlist on YouTube.

Enabling JSON schema validation in a message flow

Complete the following tasks to construct a message flow that validates a JSON document in accordance with a JSON schema:

- Enable validation at the appropriate point in the message flow, by setting the *Validate* property of the appropriate node to `Content and Value`. For more information, see "Validating messages" on page 610. If the message flow is in a REST API project, enable validation by using the REST API editor.
- Ensure that all required JSON schema files are deployed, as described in "Deploying JSON schemas" on page 561.
- Supply the name of the JSON schema that contains the definition corresponding to the JSON document. Typically, you specify this by setting the `Message model` property on a message flow node. If the parser is being invoked from a programming language such as ESQL or Java, the message model must be specified in the parameters of the function call. If the message flow is in a REST API project, the JSON schema name is set automatically.

To ensure that misspelled and unwanted JSON objects and properties are detected by validation, set the **`"additionalProperties": "false"`** property in the JSON schemas.

Deploying JSON schemas

JSON schemas with a `.json` file extension can be deployed in an application or in a library that is referenced by an application.

If the schemas are deployed as part of the application, they must be included in one of the projects that are referenced by the application.

If the schemas are deployed in a library, the application must reference that library, even if it will be deployed as part of a different application.

Standards compliant validation

JSON validation complies with the JSON schema specifications shown in the following table, with the exception of the `default`, `format`, and `discriminator` properties, which are ignored:

Specification	Definition
JSON schema draft 04	https://datatracker.ietf.org/doc/html/draft-fge-json-schema-validation-00
JSON schema draft 05	https://datatracker.ietf.org/doc/html/draft-wright-json-schema-validation-00
Swagger 2.0	https://swagger.io/specification/v2/
OpenAPI 3.0.x	https://swagger.io/specification/

Other drafts are not supported, and any attempt to use a JSON schema that specifies an unsupported schema draft or OpenAPI version results in a BIP5754 message when the deployed schema is first used.

Interpreting validation errors

A validation error occurs when the JSON document does not conform to the rules that are defined in the JSON schema. The JSON schema specifications specify exactly what these rules are, and how they should be applied. Validation errors issued by the JSON parser contain information that links the error to the construct in the JSON schema that has been violated.

All validation errors are reported in BIP5751 (parsing) or BIP5752 (writing) messages. Each message contains the following information as inserts:

- The validation error
- The location of the error in the JSON message
- The matching location in the JSON schema file
- The error context

Both locations are in the form of a JSON pointer into the document (see https://datatracker.ietf.org/doc/html/rfc6901).

For the **oneOf**, **anyOf**, **allOf**, **not**, and **dependencies** JSON schema constructs, there is an initial BIP5751 or BIP5752 message, followed by one or more BIP5751 or BIP5752 messages with the error context set to the construct name. For example, if a validation error occurs because a JSON object in the message fails to match any of the items in a **oneOf** array in the schema, an initial BIP5751 message for the **oneOf** failure is followed by a BIP5751 message for each item in the **oneOf** array, explaining why the object failed to match the item, each with error context set to **oneOf**. Nested **oneOf**, **anyOf**, and **allOf** schema constructs can result in a large number of error messages, as the JSON parser attempts to validate against all the possibilities.

If the JSON schema file does not provide enough information, you can find more information using a search engine. The JSON schema draft standards are very widely used, and many online tutorials and other resources are available.

JSON schema correctness

Deployed JSON schema are checked for well-formedness and validity by parsing them and validating them against the JSON meta-schema file for the relevant JSON schema draft or OpenAPI version. Any violations are reported when the deployed JSON schema is first used by a message flow and causes an exception to occur. If this occurs, you must correct and redeploy the JSON schema.

OpenAPI considerations

When the JSON schema is part of an OpenAPI definition, the following additional considerations apply when validation is enabled and the message flow is **not** in a REST API project:

- When the schema model is defined in the top-level OpenAPI definition, the **Message** property of the node must be specified and must be a JSON pointer to the schema model in the definition. For example:

```
"/paths/~1customers/get/responses/200/content/application~1json/schema"
```

- When the schema model is defined by a JSON schema file or OpenAPI definition that is referenced by a top-level OpenAPI definition using the `$ref` schema construct, the **Message format** property must specify the OpenAPI version that applies. Valid values are `Swagger 2.0 definition` or `OpenAPI 3.0 definition`.

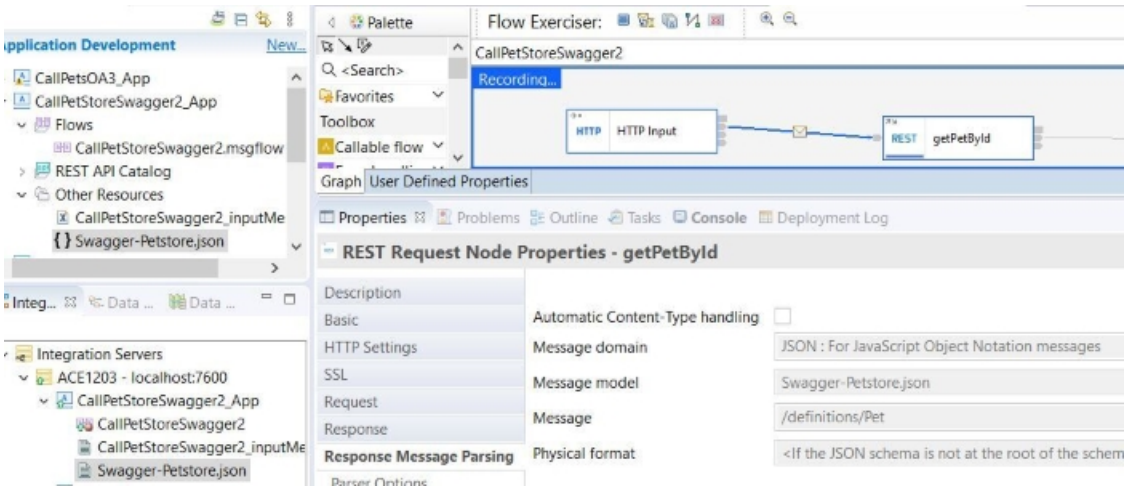
When the message flow is in a REST API project, these properties are set automatically.

The JSON models for REST API responses are typically provided in the Swagger `definitions` or OpenAPI 3 `components/schemas` section of the OpenAPI definition, and it is often simpler to specify the JSON path to that section directly. For example, the standard Pets Store Swagger 2 example operation `"getPetById"` defines its response as `"schema" : { "$ref" : "#/definitions/Pet" }`, so you can use the simple JSON pointer `"/definitions/Pet"` directly in the **Message** property on the RESTRequest node.

In the **Response Message Parsing** tab of the RESTRequest node, set the following properties:

- Deselect **Automatic Content type handling**
- Set **Message Domain** to JSON
- Set **Message model** to the OpenAPI document name
- Set **Message** to the JSON path for the relevant model

For example:



The more complex JSON pointer for this would be `"/paths/~1pet~1{petId}/get/responses/200/schema"`. When any part of the JSON path contains a `"/"`, it must be escaped as `"~1"`, which means that the `getPetById` operation's path `"/pet/{getId}"` would be `"~1pet~1{petId}"`.

Video: JSON Validation, using an OpenAPI document

This video demonstrates how you can configure a RESTRequest node to validate a JSON response message from a back-end REST API.

For more videos about JSON validation, see the IBM App Connect Enterprise playlist on YouTube.

Message tree mapping in the JSON domain
When reading a JSON message, the parser builds a message tree from the input bit stream by mapping JSON values to corresponding message tree element types. When serializing a message tree into the output bit stream, tree element types are mapped to JSON value types.

When you use the Graphical Data Mapping editor to transform a JSON message, ensure that the JSON element names that you define by using the **Add User Defined** function are valid according to the XML

element naming rules. JSON allows a wider range of characters than XML, for example a JSON object could include the @ character, however, the Graphical Data Mapping editor only allows element names that are valid in XML or XPath.

The following tables show how JSON message values are mapped to message tree element types:

- "JSON bit stream to message tree value mapping" on page 564
- "Message tree to JSON message value mapping" on page 564

JSON bit stream to message tree value mapping

The JSON parser maps JSON values to message tree element types according to the rules in the following table:

JSON value present in bit stream	Parsed as
String	CHARACTER
JSON Number value presented as: • A minus sign prefix if the value is negative • At least one digit, which can be zero • No dot fractional part • No exponent	INTEGER
JSON Number value presented as: • A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • Either or both of: • A dot and at least one digit in the fractional part, which can be zero • An exponent using uppercase 'E' or lowercase 'e', an optional plus or minus sign, and at least one digit	FLOAT
Boolean	BOOLEAN
Null	NULL

Message tree to JSON message value mapping

The JSON serializer maps message tree elements to JSON value types according to the rules in the following table:

Message tree Element type	JSON Domain serializes as	
	JSON type	Format
BIT	String	Any number of 0 and 1s
BLOB	String	Even number of hexadecimal digits

Message tree Element type	JSON Domain serializes as	
CHARACTER	String	Char data with JSON string escaping for the characters that are listed below: • " - quotation mark (U+0022) • \ - reverse solidus (U+005C) • / - solidus (U+002F) • backspace (U+0008) • form feed (U+000C) • line feed (U+000A) • carriage return (U+000D) • tab (U+0009) See https://tools.ietf.org/html/rfc7159#section-7 for more details.
DATE	String	The standard ESQL string representation, 'yyyy-mm-dd'
TIME, GMTTIME	String	The standard ESQL string representation, 'hh:mm:ss.ffffff'
INTEGER	Number	• A minus sign prefix if the number is negative • At least one digit, which can be zero • No dot fractional • No exponent
FLOAT	Number	• A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • At least one digit in the fractional part, which can be zero • An exponent part using uppercase 'E', a plus or minus sign, and at least one digit
DECIMAL	Number	• A minus sign prefix if the number is negative • At least one digit in the integer part, which can be zero • At least one digit in the fractional part, which can be zero Decimal literals ‘NAN', ‘INF', and so on, are not supported when serializing to JSON.
BOOLEAN	Boolean	true or false The serializer only serializes Boolean logical tree elements with true or false values; unknown is not supported
NULL	Null	Null

Message tree Element type	JSON Domain serializes as	

ROW	Object	Note: Assigning a ROW directly to a JSON Domain tree does not produce JSON arrays.
		`DECLARE myRow ROW;`
		`SET myRow.rowData[1].fieldOne = 'Row1Field1';`
		`SET myRow.rowData[1].fieldTwo = 'Row1Field2';`
		`SET myRow.rowData[2].fieldOne = 'Row2Field1';`
		`SET myRow.rowData[2].fieldTwo = 'Row2Field2';`
		`SET OutputRoot.JSON.Data.aRow = myRow;`
		Produces the following JSON bit stream
		`"aRow":{`
		`"rowData":`
		`{"fieldOne":"Row1Field1","fieldTwo":"Row1Field2"},"`
		`"rowData":`
		`{"fieldOne":"Row2Field1","fieldTwo":"Row2Field2"}`
		`}`

ROW	Array	To produce a JSON array from a ROW type the JSON.Array field would also be set.
		`SET OutputRoot.JSON.Data.aRow = myRow;`
		`SET OutputRoot.JSON.Data.aRow TYPE = JSON.Array;`
		Produces the following JSON bit stream
		`"aRow":[`
		`{"fieldOne":"Row1Field1","fieldTwo":"Row1Field2"},`
		`{"fieldOne":"Row2Field1","fieldTwo":"Row2Field2"}`
		`]`

Handling null values

A business message might contain fields that are either empty or have a specific out-of-range value. In these cases, the application that receives the message is expected to treat the field as if it did not have a value. The logical message tree supports this concept by enabling the value of any element to be set to NULL.

Ways to represent a null value

In an XML document, the usual way to represent a null value is to leave the element or attribute empty.

For example: `<price></price>` or `<element price=""/>`

The xsi:nil attribute provides a way to make this more explicit: `<price=xsi:nil="true"/>`

Some business messages use a special value to represent a null value: `<price>-999</price>`. This style of null representation is supported by the DFDL and MRM parsers.

DFDL parser support for null values

The DFDL parser can detect a null value that is represented by an out-of-range value. The null value must be specified in the DFDL schema, and can be the same or different for each element. In DFDL it is called a *nil value*. While parsing, the DFDL parser checks the nil value for each element in the message. If the value in the bit stream matches the nil value in the DFDL schema, the DFDL parser sets the value in the message tree to NULL. The same check is performed when converting a message tree to a bit stream. If the value in the message tree is NULL, the DFDL parser outputs the nil value from the DFDL schema.

For more information, see The DFDL 1.0 specification, section 13.15.

Graphical Data Mapper support for null values

In the Graphical Data Mapping editor, you can create an element in the message tree as NULL, you can set the value of a message tree element to NULL, and you can use XPath expressions in your graphical transformations to test if an input element is NULL.

When you map null values, consider the behavior of the Graphical Data Mapping editor. For more information, see Handling nulls in message maps.

When you test to check if an input element is NULL, you can use any of the following XPath expressions: **fn:empty**, **fn:nilled**, or **fn:exists**. For more information, see Choosing an XPath conditional expression that tests for a nil value in a transform .

ESQL support for null values

Using ESQL, you can set the value of a message tree element to NULL:

```
SET OutputRoot.XMLNSC.myField VALUE = NULL;
```

Note that this is different from `SET OutputRoot.XMLNSC.myField = NULL;` which would cause myField to be deleted from the message tree.

The same effect can be achieved using Java.

XMLNSC parser support for null values

Typically, the XML parsers (XMLNSC, XMLNS, and XML) do not create null values in the message tree; an empty element or an empty attribute value merely produces an empty string value in the message tree.

If validation is enabled, the XMLNSC parser detects and processes any xsi:nil attributes in the input document. If the xsi:nil attribute is set to 'true', and the element is nullable, the attribute's parent element in the message tree is given a null value.

For more information about XML parser support for empty elements and null values, see “XMLNSC empty elements and null values” on page 532 and “XMLNS empty elements and null values” on page 541.

JSON parser support for null values

The JSON format supports null as a JSON value type for an object.

When a JSON message includes an object with a null value, the JSON parser sets the value in the message tree to NULL.

When the JSON serializer includes an element in the message tree with a NULL value, the JSON bit stream is constructed as a JSON object with a value of null.

MRM parser support for null values

XML physical format

When parsing, the MRM XML parser can detect and process xsi:nil attributes in the input XML document. If the xsi:nil attribute is set to 'true', and the element is nullable, the attribute's parent element in the message tree is given a null value.

For information about enabling xsi:nil support in the MRM parser, see Message Sets: XML Null handling options.

The following topics provide more information about handling null values in the MRM parser:

- “MRM Custom wire format: NULL handling” on page 2182
- “MRM XML physical format: NULL handling” on page 2202
- “MRM TDS format: NULL handling” on page 2197

All physical formats

The MRM parser can detect a null value that is represented by an out-of-range value. The null value must be specified in the physical format of the message set.

While parsing, the MRM parser checks the null value for each element in the message. If the value in the bit stream matches the null value in the message set, the MRM parser sets the value in the message tree to NULL.

The same check is performed when converting a message tree to a bit stream. If the value in the message tree is NULL, the MRM parser outputs the null value from the message set.

Properties

You can view and change properties that define integration node characteristics, and those properties of associated resources such as message flows.

Integration node properties
Each integration node has a set of properties that define certain characteristics that you can control:

- You can set some integration node properties only at the time that you create the integration node by using the **mqsicreatebroker** command; for example, its associated queue manager.
- You can access some integration node properties from ESQL and Java programs that you run in message flow nodes.

Message flow node properties
Each built-in (supplied) node has at least one property that you can configure; some properties are mandatory, others are optional. In addition to setting these properties for each node when you add it to a message flow, you can also use the following configuration techniques:

- You can promote properties, so that you can set them at the message flow level. For information about how and why you might want to use this technique, see "Promoted properties" on page 568.
- You can configure properties when you add the message flow to a BAR file for deployment. In each node description, the subset of node properties that you can use in this way are identified.
- You can override some of the properties that are set on message flow nodes by using policies (see "Overriding properties at run time with policies" on page 324).

Node properties are discussed in "Message flow nodes" on page 484.

User-defined properties
You can create your own properties when you create a message flow, and access those properties from ESQL and Java programs in your message flow nodes. For further details, see "User-defined properties" on page 569

Promoted properties

A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

A message flow contains one or more message flow nodes. You can promote the properties of a message flow node to apply to the message flow to which it belongs. In this case, any user of the message flow can override these promoted properties at the message flow level, without being aware of the message flow's internal structure.

You can promote compatible properties (that is, properties that represent comparable values) from more than one node to the same promoted property; you can then set a single property that affects multiple nodes.

For example, you might want to set the name of a data source as a property of the message flow, rather than a property of each individual node in the message flow that references that data source. You create a message flow that accesses a database called SALESDATA. However, while you are testing the message flow, you want to use a test database called TESTDATA. If you set the data source properties of each individual node in the message flow to refer to SALESDATA, you can promote the data source property for each node in the flow that refers to it, and update the property to have the value TESTDATA; this value overrides the data source properties on the nodes while you test the message flow (the promoted property always takes precedence over the settings for the properties in any relevant nodes).

A subset of message flow node properties is also configurable (that is, the properties can be updated at deployment time). You can promote configurable properties: if you do so, the promoted property (which

can have a different name from the property or properties that it represents) is the one that is available to update at deployment time. Configurable properties are those associated with system resources; for example, queues and data sources. An administrator can set these properties at deployment time, without the need for a message flow developer.

User-defined properties

A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. This property can be used by the Compute, JavaCompute, .NETCompute, and Mapping nodes.

The advantage of UDPs is that their values can be changed by operational staff at deployment and run time. You do not need to change your application programs. For example, if you use the UDPs to hold data about your computer center, you can configure a message flow for a particular computer, task, or environment at deployment time, without having to change the code at the message node level.

When you launch the Message Flow editor to either create a message flow or modify an existing message flow, as well as deciding which nodes are required in the message flow, you also have the option (provided by the tab) of defining and giving initial values to some user-defined properties. Use the **User Defined Properties** tab at the bottom of the edit window. See Message Flow editor for more information. You can also define a UDP at the level of the integration server, by updating the `server.conf.yaml` configuration file.

You can access UDPs in the following ways:

- From a Mapping node within a Graphical Data Map. Use the `iib:getUserDefinedProperty("propertyname")` custom XPath function; see Accessing user-defined properties from a Mapping node. You can also access UDPs from custom ESQL that is invoked from a map.
- From a JavaCompute node, by calling the `getUserDefinedAttribute` method.
- From a Compute node, by using a DECLARE statement with the EXTERNAL keyword, which can also provide a default.
- From a .NETCompute node, by calling the `GetUserDefinedProperty` method.

See the DECLARE statement for details of the DECLARE statement, and see “Accessing message flow user-defined properties from a JavaCompute node” on page 1784 for more information about how to use a UDP in a JavaCompute node.

If you use ESQL to access your UDP, the value that you give to a UDP when you define it in a message flow overrides the value of that variable in your ESQL program.

You can also modify the value of a UDP at deployment time by using the Broker Archive editor to edit the BAR file. This value overrides the value that was set when you defined the message flow.

You can also change the behavior of a message flow at run time by using the administrative REST API to modify the values of user-defined properties on the message flow. For more information, see “Setting message flow user-defined properties at run time by using the administration REST API” on page 436.

The value of the UDP is set at the message flow level, and is the same for all eligible nodes that are contained in the flow. An *eligible node* is a node that supports UDPs and is within the scope of the declaration that declares the UDP to your application. For example, if you use the Message Flow editor to change the value of a user property called `timezone`, which is declared in a schema called *mySchema*, in a message flow called *myFlow*, the UDP is available at run time to all the nodes in *myFlow* that support UDPs and that fall within *mySchema*.

- If you use the Message Flow editor to change the value of a user-defined property in a subflow, the newly edited property is available to all the nodes in the subflow that support UDPs, and that are within the scope of the declaration. The property is not available, for example, to nodes in the parent flow.
- If a node within a subflow accesses a UDP and if the subflow does not define the UDP, the parent flow is searched for a definition of the UDP at run time. If the parent flow does not define the UDP and it is also a subflow, the parent of this subflow is searched for a definition of the UDP. The search process

continues in this recursive manner upwards through the chain of subflows and parent flows until the UDP definition is found, or until the top-level flow is reached.

To have a value that is available to both the main and subflow, define UDPs in one of the following ways:

- Define UDPs in the subflow and then promote them to the main flow. Set the value in the main flow.
- Define UDPs in the main flow only and allow the subflow to locate the UDP value in the main flow at run time.

Controlling user-defined properties at run time

User-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the administration REST API to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

For example, a message flow contains a Route node, which is used to differentiate between the classes of customer that are defined in the message. The Route node has a user-defined property called `ProcessClasses`, which is set with an initial value of `All`. When `ProcessClasses` is set to `All`, the node routes messages from all classes of customer to its first terminal for immediate processing.

When certain conditions are detected (for example, the monitoring system detects that the request load is causing the service level agreement to fall below its target), the Route node must be set to pass requests from only "Gold" class customers for immediate processing, while other customer requests are sent to another output terminal, which queues them for later batch processing. Therefore, the monitoring application sets `ProcessClasses` to `Gold`, so that the Route node routes the less critical messages to the second terminal.

To make it easier to know what a user-defined property does, and what values it can have, adopt a suitable naming convention. For example, a user-defined property named `property01`, with an initial value of `valueA` is not as useful as a property named `RouteToAorB` with an initial value of `RouteA`.

For information about how to use the REST API to control user-defined properties at run time, see “Setting message flow user-defined properties at run time by using the administration REST API” on page 436. For more information about the administration REST API, see “Managing resources by using the administration REST API” on page 334.

Precedence of UDP value overriding

You can define a user-defined property in the following ways:

- In the ESQL code
- In the Message Flow editor
- Through a BAR file override before BAR file deployment
- By using the administration REST API

A BAR file override takes precedence over changes in the Message Flow editor, and changes in the Message Flow editor take precedence over changes in the ESQL code.

The precedence of the values for user-defined properties is shown in the following sequence:

1. The user-defined property `ProcessClasses` is set to `All` in a message flow BAR file. After deployment of the BAR file, the value of `ProcessClasses` is `All`.
2. A user-defined property override is set to `Gold` by using the administration REST API. After successful completion of this method, the value of `ProcessClasses` is `Gold`.
3. The integration node or server is shut down and restarted. The value of `ProcessClasses` reverts to `All` because the REST API update is not persistent.
4. A user-defined property override is set to `Gold` by using the administration REST API. After successful completion of this method, the value of `ProcessClasses` is `Gold`.
5. The original flow BAR file is redeployed. After deployment, the value of `ProcessClasses` is `All`.

Data conversion

Convert data that your message flows are transferring between different environments by using IBM MQ or IBM App Connect Enterprise facilities.

Data conversion is the process by which data is transformed from the format that is recognized by one operating system into the format that is recognized by a second operating system with different characteristics such as numeric order.

If you are using a network of systems that use different methods for storing numeric values, or you need to communicate between users who view data in different code pages, you must consider how to implement data conversion.

Code page conversions

Code page conversion might be required for one or more of the following reasons:

- ASCII versus EBCDIC
- Code pages that are specific to national language
- Code pages that are specific to operating systems

In IBM MQ, these factors are handled by the CCSID field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the IBM MQ product documentation online. For more information about code page support, see "Code page conversion", also in the *Application Programming Reference* section.

Encoding

Encoding (byte order) conversion might be required for one or both of the following reasons:

- Big endian versus little endian

  Endian is an attribute of data that describes whether it is stored in computer memory or transmitted with the most significant byte first (big endian) or last (little endian).
- Floating point number representations

In IBM MQ, these factors are handled by the Encoding field in the MQMD header. For more information about the MQMD header, see "MQMD - Message descriptor" in the *Application Programming Reference* section of the IBM MQ product documentation online. For more information about encoding, see "Machine encoding", also in the *Application Programming Reference* section.

If you are configuring a message flow to receive messages:

- Messages received across an IBM MQ protocol that uses IBM MQ headers, contain code page encoding characteristics in the MQMD header, and optionally in other IBM MQ headers.
- Messages received across protocols that do not use IBM MQ headers do not include these characteristics. Configure these characteristics by using properties on the nodes in your message flows. For example, set the Message coded character set ID and Message encoding properties on the FileInput node.

If you are configuring a message flow to send messages to other applications or systems:

- Messages sent across an IBM MQ protocol contain code page encoding characteristics in the MQMD header, and optionally in other IBM MQ headers.
- Messages sent across protocols that do not use IBM MQ headers must be modified to include these characteristics in the Properties folder in the logical message tree structure. The parser called by the output node uses these values to generate the correct bit stream.

When you use IBM App Connect Enterprise, you can use the data conversion facilities of IBM App Connect Enterprise, IBM MQ, or both.

IBM App Connect Enterprise facilities

You can model your messages in the MRM domain through the IBM App Connect Enterprise Toolkit. Predefined elements of the messages are converted according to their type and physical layer characteristics. For more information, see “Message Sets: Configuring physical properties” on page

2287. You can also use self-defining messages. You can then use the Compute, or JavaCompute node to configure encoding and CCSIDs. You do not need IBM MQ data conversion exits.

- String data is converted according to the CCSID setting.
- Decimal integer and float extended decimal types are converted according to the CCSID setting.
- Decimal integer and float (other physical data types) are converted according to the Encoding setting.
- Binary and Boolean data is not converted.

IBM App Connect Enterprise can also convert the IBM MQ headers for which parsers are provided.

When you use IBM App Connect Enterprise facilities, the whole message is not converted to the specified encoding and CCSID: you can specify a different encoding, or CCSID, or both, in each header to perform a different conversion for the following part of the message. The encoding and CCSID in the last header defines the values for the message body.

IBM MQ facilities

Headers and message body are converted according to the values set in the appropriate MQMD fields, and other header format names. You might need to set up data conversion exits to convert the body of your messages.

When you use IBM MQ facilities, the whole message is converted to the specified encoding and CCSID, according to the setting of the format in the IBM MQ header.

For more detail about data conversion using IBM MQ facilities, see "Data conversion" in the *Application Programming Reference* section of the IBM MQ product documentation online.

Managing message flow resources

You can use applications, libraries, and integration projects to organize your resources.

Before you begin

- To learn about the uses and benefits of applications and libraries, see “Benefits of using applications and libraries” on page 1959.
- To learn more about integration projects, see “Integration projects” on page 1959.

About this task

You can use applications, libraries, and integration projects to build your solution. An application is a container for everything that you need for your solution. A library contains a set of resources that you might want to reuse for other solutions. You can refer to one or more libraries from an application. You can deploy your solution by dragging the appropriate resources onto an integration server, or by adding them to a BAR file.

Message flow projects have been replaced by integration projects. When you import resources from WebSphere Message Broker Version 7.0, message flow projects are converted automatically to integration projects. You might want to convert your integration projects to applications or libraries, depending on the way in which you intend to use these resources.

You can choose the order in which you create your resources and their containers. For example, you can create a library before or after you create an application. The following section suggests a possible order in which to complete these tasks, depending on whether you are importing resources from a previous version or creating resources from scratch.

- “Starting by creating an application” on page 573
- “Converting existing resources to applications and libraries” on page 573
- “Organizing resources” on page 573

Starting by creating an application

About this task

The following steps describe how to build a solution by creating a container and filling it with resources.

Procedure

1. Create an application by following the instructions in "Creating an application" on page 1963.
2. Optional: Create a library by following the instructions in "Creating a library" on page 1964.
3. Create a message flow in your application by following the instructions in "Creating a message flow" on page 574.
4. You can create a default broker schema when you create the message flow, but to create another broker schema, follow the instructions in "Creating a broker schema" on page 1966.
5. Create other resources that you need in the application or library by following the instructions in "Creating resources in an application" on page 1964 or "Creating resources in a library" on page 1965.
6. Refer to one or more libraries of resources from the application, as described in "Referencing resources in other libraries" on page 1976.
7. Deploy your solution by using one of the following methods.
   - Add your applications, libraries, and other required resources to a BAR file, and deploy the BAR file.
   - Deploy your resources by dragging them from the Application Development view onto an integration server. If your solution includes shared libraries, deploy them to the integration server before you deploy the applications that refer to them.

   For more information, see Chapter 7, "Deploying integration solutions," on page 2463.

Converting existing resources to applications and libraries

About this task

You can import resources from previous versions of IBM App Connect Enterprise and use them as a basis for an application or library.

Procedure

1. Export a project from a previous version in a Project Interchange file.
2. Import the Project Interchange file to Version 12.0.
3. Convert or add the project to an application or library by following the instructions in the appropriate topic:
   - "Adding a project to an application, integration service, or library" on page 1979

Organizing resources

About this task

When you have created your resources, the following topics explain how to organize them.

- "Opening an existing message flow or subflow" on page 577
- "Copying a message flow or subflow by using copy" on page 580
- "Renaming a message flow or subflow" on page 578
- "Moving a message flow or subflow" on page 578
- "Deleting a message flow or subflow" on page 579
- "Deleting a schema" on page 1983
- "Adding version information and custom keyword values to your development resources" on page 2466

- "Saving a message flow or subflow" on page 581

Creating a message flow

Create a message flow to specify how to process messages in the integration node. You can create one or more message flows and deploy them to one or more integration nodes.

Before you begin

- You can create an application, library, or integration project at the same time as creating a message flow, or you can create it before you create a message flow. To create a container first, follow the instructions in the following topics:
  - "Creating an application" on page 1963
  - "Creating a library" on page 1964
  - "Creating an integration project" on page 1969
- One of the steps involved in creating a message flow is to select an existing broker schema, or create a new one. To use a particular broker schema with your message flow, create the schema by following the instructions in "Creating a broker schema" on page 1966.

About this task

The message flow and its resources are stored in a file system or in a shared repository. A file system can be on the local drive, or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

Use this process to create a `.msgflow` file, in which you can define either a complete message flow that you can deploy, or a subflow that provides a subset of function (for example, a reusable error processing routine), that you cannot deploy on its own. To create a subflow that you can deploy as an individual resource, you must define the subflow in a `.subflow` file. For more information about subflows and the differences between subflows that are defined in `.subflow` files and subflows that are defined in `.msgflow` files, see "Subflows" on page 488.

The mode in which IBM App Connect Enterprise is running can affect the number of message flows that you can use; see "Restrictions that apply in each operation mode" on page 5.

To create a message flow, complete the following steps.

Procedure

1. To open the New Message Flow wizard, click **File** > **New** > **Message Flow**.

   Alternatively, in the Application Development view, right-click an application, library, or integration project and click **New** > **Message Flow**.
2. Specify the application, library, or integration project in which to create the message flow. Either select an existing container from the list, or create a container by clicking **New**.

   If you right-clicked a particular container in step "1" on page 574, the name of that container is selected automatically.
3. In the Message flow **Name** field, enter a name for the new message flow.

   You can use all valid characters for the name, but it is helpful to choose a name that reflects its function, for example, OrderProcessing.
4. Decide whether you want to create a default broker schema.

   By default, a default broker schema is created. If you do not want to create a default schema, clear the check box and select an existing schema from the list. However, if no schemas exist in the project, you must create a default schema at this point.
5. Click **Finish**.

Results

The new message flow (*message_flow_name*.msgflow) is displayed in the Application Development view, under the application, library, or integration project. The message flow opens in the Message Flow editor so that you can add some message flow nodes.

What to do next

Define the content of the message flow by following the instructions in "Defining message flow content" on page 614.

Creating a subflow

Create a .subflow file, in which you can define subflow content to include in your message flow applications. Use .subflow files to create subflows that you can deploy to an integration server as individual resources, independent of the message flows that use the subflow. You can create one or more subflows and deploy them to one or more integration servers.

Before you begin

- You can create an application, library, or integration project at the same time as creating a subflow, or you can create the container before you create the subflow. To create a container first, follow the instructions in the following topics:
  - "Creating an application" on page 1963
  - "Creating a library" on page 1964
  - "Creating an integration project" on page 1969

  When an application or shared library references other shared libraries, all the subflows for a broker schema must be in a single container. Subflows for a broker schema must not be in both an application (or shared library) and a shared library that is referenced by that application (or shared library). Subflows for a broker schema must not be in two or more shared libraries that are referenced by a single application or shared library. All the subflows in a broker schema must be either in the main application or shared library, or in a single referenced shared library.
- One of the steps involved in creating a subflow is to select an existing broker schema, or create a new one. Subflows in shared libraries cannot be in the default broker schema. To use a particular broker schema with your subflow, create the schema by following the instructions in "Creating a broker schema" on page 1966.

About this task

You can create subflows to define content that provides a subset of function (for example, a reusable error processing routine) that you can reuse in your message flow applications. You can create subflows either in .subflow files or in .msgflow files. For more information about subflows and the differences between subflows that are defined in .subflow files and subflows that are defined in .msgflow files, see "Subflows" on page 488. The information in this topic describes how to create a .subflow file.

The subflow and its resources are stored in a file system or in a shared repository. A file system can be on the local drive, or a shared drive. If you store files in a repository, you can use all the available repositories that are supported by Eclipse; for example, CVS.

The mode in which IBM App Connect Enterprise is running can affect the number of message flows that you can use; see "Restrictions that apply in each operation mode" on page 5.

To create a .subflow file, complete the following steps:

Procedure

1. To open the New Subflow wizard, click **File** > **New** > **Subflow**.

   Alternatively, right-click an application, library, or integration project and click **New** > **Subflow**.

2. Specify the application, library, or integration project in which to create the subflow. Either select an existing container from the list, or create a container by clicking **New**.

   If you right-clicked a particular container in step "1" on page 575, the name of that container is selected automatically.
3. In the **Subflow name** field, enter a name for the new subflow.

   You can use all valid characters for the name, but it is helpful to choose a name that reflects its function, for example, ErrorProcessing.
4. Decide whether you want to create a default broker schema.

   By default, a default broker schema is created. Subflows in shared libraries cannot be in the default broker schema. If you do not want to create a default schema, clear the check box and select an existing schema from the list. However, if no schemas exist in the project, you must create a schema at this point.
5. Click **Finish**.

Results

The new subflow file (*subflow_name*`.subflow`) is displayed in the Application Development view, under the application, library, or integration project. The subflow opens in the Message Flow editor so that you can define the subflow content by adding, configuring, and connecting message flow nodes. An Input node and an Output node are added to the subflow.

What to do next

1. Define the content of the subflow by following the instructions in "Defining message flow content" on page 614. Every Input node defined in the subflow represents an In terminal on the subflow node. Every Output node defined in the subflow represents an Out terminal on the subflow node. You can use these terminals to connect the subflow node to other nodes in the parent flow. You cannot add the following nodes to a subflow that is defined in a `.subflow` file:
   - Nodes representing subflows that are defined in `.msgflow` files
   - User-defined nodes created from subflows that are defined in `.msgflow` files
2. Add your subflow to a message flow; see "Adding a subflow" on page 623.

Converting between message flows and subflows

Convert a `.msgflow` file to a `.subflow` file, or a `.subflow` file to a `.msgflow` file.

Before you begin

This task assumes that you have already created a message flow or subflow; for more information, see "Creating a message flow" on page 574 or "Creating a subflow" on page 575.

About this task

You can convert `.msgflow` files to `.subflow` files, or `.subflow` files to `.msgflow` files.

When you convert `.msgflow` files to `.subflow` files, the following referenced files must be updated. You can complete these updates when you convert your `.msgflow` file.

- If the `.msgflow` file is used as a subflow, the parent flow must be updated so that it references the new `.subflow` file.
- If the `.msgflow` file contains subflows that are defined in `.msgflow` files, these subflows must also be converted to `.subflow` files.

Procedure

To convert a `.msgflow` file or a `.subflow` file, complete the following steps:

1. In the Application Development view, expand the appropriate application, library, or integration project.
2. To convert a `.msgflow` file to a `.subflow` file, complete the following steps:
   a) Right-click the message flow that you want to convert and click **Convert to subflow**.
   b) If the message flow is used as a subflow or contains subflows, the **Convert message flow to subflow** window opens and shows referenced files that must be updated. Select the files you want to update and click **OK**.

      The subflow is saved and its contents are validated. The editor provides a report of errors and warnings in the Problems view.
3. To convert a `.subflow` file to a `.msgflow` file, complete the following steps:
   a) Right-click the subflow that you want to convert and click **Convert to Message Flow**.
   b) Specify the name for the flow.

      You cannot use the name of a file that already exists in that application, library, or integration project. You must include `.msgflow` at the end of your file name.
   c) Click **OK**.

      The message flow is saved and its contents are validated. The editor provides a report of errors and warnings in the Problems view.

What to do next

For information about handling errors that occur when you save a message flow or subflow, see "Correcting errors from saving a message flow or subflow" on page 583.

Opening an existing message flow or subflow

Open an existing message flow or subflow to change or update its contents, or to add or remove nodes.

Before you begin

This task assumes that you have already created a message flow or subflow; for more information, see "Creating a message flow" on page 574 or "Creating a subflow" on page 575.

About this task

The Application Development view is populated with all the applications, libraries, and integration projects to which you have access. Message flows and subflows are contained in these containers. Message flow files are called *message_flow_name*`.msgflow`. Subflow files are called *subflow_name*`.subflow`.

Message flow projects have been replaced by integration projects in WebSphere Message Broker Version 8.0.

To open an existing message flow or subflow, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project.
2. Double-click the message flow or subflow that you want to open.

   The graphical view of the message flow is displayed in the Message Flow editor. You can now work with this message flow; for example, you can add or remove nodes, change connections between nodes, or modify node properties.
3. Some message flow nodes use data from associated files. For example, a Compute node has an associated ESQL file, a Mapping node has an associated map file, and a JavaCompute node has an associated Java file. You can open these associated files by double-clicking the node.

4. When you have completed your changes, save the message flow or subflow.

Renaming a message flow or subflow

You can rename a message flow or subflow. You might want to rename a flow if you have modified it to provide a different function, and you want the name of the flow to reflect this new function.

Before you begin

This task assumes that you have created a message flow or subflow. For more information, see “Creating a message flow” on page 574 or “Creating a subflow” on page 575.

About this task

To rename a message flow or subflow, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project.
2. To open the Rename Resource dialog box, select the flow that you want to rename (*message_flow_name*.msgflow or *subflow_name*.subflow), then click **File** > **Rename**.
3. Enter a new name for the file.
4. Click **OK** to rename the file, or **Cancel** to cancel the request.

   If you click **OK**, the file is renamed.

   After you have renamed the flow, any references that you have to this flow (for example, if it is embedded in another message flow) are no longer valid.

Moving a message flow or subflow

You can move a message flow or subflow from one application, library, or integration project to another. You might want to move a flow, for example, if you are reorganizing the resources in your projects.

Before you begin

This task assumes that you have already created a message flow or subflow; for more information about creating a message flow, see “Creating a message flow” on page 574. For more information about creating a subflow, see “Creating a subflow” on page 575.

About this task

The Application Development view is populated with all the applications, libraries, and integration projects to which you have access. Message flows and subflows are contained in these containers. Message flow files are called *message_flow_name*.msgflow. Subflow files are called *subflow_name*.subflow.

To move a message flow or subflow, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project, then expand the broker schema folder.

2. Right-click the message flow or subflow that you want to move, click **Move**, then select the destination application, library, or integration project.

   Alternatively, you can drag the flow to a new location.

   If the location to which you have dragged a flow is not valid, a no entry symbol is displayed, and the message flow or subflow is not moved.

   If you have created an empty broker schema for this purpose, it might not be visible in the Application Development view if category mode is selected. To see an empty schema in the Application Development view, click **Hide Categories**  .
3. Check the Problems view for errors or warnings that are generated by the move.

   Errors are indicated by the error icon  and warnings are indicated by the warning icon  . The Problems view includes errors that are caused by integration node references. When the move is complete, all references to this flow (for example, if this message flow is a reusable error routine that you have embedded in another message flow) are checked.
4. Double-click each error or warning to correct it.

   The message flow or subflow that contains the error is opened in the editor view and the node in error is highlighted.

Results

When you move a message flow or subflow, the associated files (for example, all ESQL or mapping files) are not automatically moved to the target broker schema. To move these files as well, you must follow the procedure in this topic for each file.

Deleting a message flow or subflow

Delete message flows or subflows from your application, library, or integration project when you no longer need them.

Before you begin

Deleting a message flow in the IBM App Connect Enterprise Toolkit deletes the project and its resources. If you are using a shared repository, the repository might retain a copy of a deleted resource.

This task assumes that you have created a message flow or subflow. For more information, see “Creating a message flow” on page 574 or “Creating a subflow” on page 575.

About this task

The Application Development view is populated with all the applications, libraries, and integration projects to which you have access. Message flows and subflows are contained in these containers. Message flow files are called *message_flow_name*`.msgflow`. Subflow files are called *subflow_name*`.subflow`. Applications and libraries are listed at the top level of the Application Development view; integration projects are listed in the Independent Resources folder.

To delete a message flow or subflow, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project.
2. Select the message flow or subflow that you want to delete, then click **Edit** > **Delete**.

   A confirmation dialog box is displayed.

3. Click **Yes** to delete the message flow or subflow, or **No** to cancel the delete request.

   When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.
4. Check the Problems view to see if errors have been returned to the delete request.

   Errors are generated if you delete a message flow or subflow that is embedded in another flow, because the reference is no longer valid.

   a) Click the error in the Problems view.

      The flow that now has a non-valid reference is displayed.

   b) Either remove the node that represents the deleted flow from the parent flow, or create another message flow or subflow with the same name to provide whatever processing is required.

Results

When you delete the message flow or subflow, the files that are associated with the flow (the ESQL and mapping files, if present) are not deleted by this action. If you want to delete these files also, you must do so explicitly.

Copying a message flow or subflow by using copy

Copy a message flow or subflow to use as a starting point for a new message flow or subflow that has similar function. For example, you might want to replace or remove one or two nodes to process messages in a different way.

Before you begin

To complete this task you must have already created a message flow or subflow; for more information, see "Creating a message flow" on page 574 or "Creating a subflow" on page 575.

About this task

The Application Development view is populated with all the applications, libraries, and integration projects to which you have access. Message flows and subflows are contained in these containers. Message flow files are called *message_flow_name*`.msgflow`. Subflow files are called *subflow_name*`.subflow`.

To copy a message flow, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project.
2. Select the message flow or subflow that you want to copy and click **Edit** > **Copy**.

3. Select the application, library, or integration project to which you want to copy the message flow or subflow, then click **Edit** > **Paste**.

   You can copy the flow to a different container, or to the same container. If the target container does not contain a broker schema, the schema is created automatically when you paste the message flow or subflow. If you copy a flow to the same container, you are prompted to rename the flow.

   When you copy a message flow or subflow, the associated files (ESQL and mapping, if present) are not automatically copied to the same target container. To copy these files as well, you must copy and paste each file.

   You might also need to update nodes that have associated ESQL or mappings, to ensure that modules are unique. For example, you have created a message flow Test1 that contains a single Compute node. You copy message flow Test1 and its associated `.esql` file to the same application, library, or integration project, and rename the copy Test2. However, two modules named Test1_Compute now exist in the same schema: one is in `Test1.esql` and the second is in `Test2.esql`.

   This duplication is not supported, and an error message is written to the Problems view when you have completed the copy action. You must rename the associated ESQL modules in the `.esql` file and update the matching node properties to ensure that every module in a broker schema is unique.

Results

When you have completed these steps, the flow is copied with all property settings intact. If you intend to use this copy of the message flow or subflow for another purpose (for example, to retrieve messages from a different input queue), you might have to modify its properties.

You can also copy a flow by clicking **File** > **Save As**. This task is described in "Saving a message flow or subflow" on page 581.

Saving a message flow or subflow

You might want to save your message flow or subflow when you close the IBM App Connect Enterprise Toolkit, work with another resource, or validate the contents of the flow.

Before you begin

This topic assumes that you have created or opened a message flow or subflow, as described in the following topics:

- "Creating a message flow" on page 574
- "Creating a subflow" on page 575
- "Opening an existing message flow or subflow" on page 577

About this task

To save a message flow or subflow, complete the following steps.

Procedure

1. To save the flow without closing it, click **File** > **Save**.

   You can also save everything by clicking **File** > **Save All**.

   The message flow or subflow is saved and the message flow validator validates the contents of the flow. The validator reports all errors that it finds in the Problems view. The flow remains open in the editor view. For example, if you save a message flow and have not set a mandatory property, an error message appears in the Problems view and the editor marks the node with the error icon  . The message flow in the Application Development view is also marked with the error icon. This error can

occur if you have not edited the properties of an MQInput node to define the queue from which the input node retrieves its input messages.

You might also get warnings when you save a message flow or subflow; warnings are also shown in the Problems view. Warnings indicate that, although the configuration of the message flow or subflow does not contain an explicit error, the configuration might result in unexpected results when the flow completes. For example, if you have included an input node in your message flow that you have not connected to another node, you get a warning.

2. If you save a message flow or subflow that includes a subflow, and the embedded subflow is no longer available, three error messages are added to the Problems view.

   These errors indicate that the input and output terminals and the embedded subflow itself cannot be located. This error can occur if the embedded subflow has been moved or renamed.

   To resolve this situation, right-click the subflow node in error and click **Locate Subflow**. The Locate Subflow dialog box is displayed, listing the available integration projects. Expand the list and explore the resources available to locate the required subflow. Select the correct subflow and click **OK**. All references in the current message flow or subflow are updated and the errors are removed from the Problems view.

3. You can also save a message flow or subflow when you close it. When you click **File** > **Close**, you are asked if you want to save the flow. Click **Yes** to save and close.

   Validation occurs and all errors and warnings are written to the Problems view.

What to do next

For information about using the **File** > **Save As** option to take a copy of the current flow, see "Copying a message flow or subflow by using the save option" on page 582.

For information about handling errors that occur when you save, see "Correcting errors from saving a message flow or subflow" on page 583.

Copying a message flow or subflow by using the save option

You can copy a message flow or subflow by using the **File** > **Save As** option.

Before you begin

This topic assumes that you have created or opened a message flow or subflow, as described in the following topics:

- "Creating a message flow" on page 574
- "Creating a subflow" on page 575
- "Opening an existing message flow or subflow" on page 577

Procedure

1. Click **File** > **Save As**.
2. Specify the application, library, or integration project in which you want to save a copy of the message flow or subflow.

   By default, the current project is selected; you can accept this name, or choose another name from the list.

3. Specify the name for the new copy of the flow.

   To save this message flow or subflow in the same project, you must either give it another name, or confirm that you want to overwrite the current copy.

   To save the message flow or subflow in another project, the project must already exist. You can save the flow with the same name or another name in another project.

4. Click **OK**.

   The message flow or subflow is saved and its contents are validated. The editor provides a report of errors and warnings in the Problems view.

What to do next

For information about handling errors that occur when you save a message flow, see "Correcting errors from saving a message flow or subflow" on page 583.

Correcting errors from saving a message flow or subflow

Correct the errors that are reported when you save a message flow or subflow.

About this task

When you save a message flow or subflow, as described in "Saving a message flow or subflow" on page 581, the flow is validated. Errors and warnings are shown in the Problems view. The following steps describe how to correct errors that occur when you save a flow.

Procedure

1. Examine the list of errors and warnings in the Problems view.
2. Double-click each entry in turn.

   The message flow or subflow is opened in the editor view (if it is not already open), and the editor selects the node in which the error was detected. If the error has been generated because you have not set a mandatory property, the editor also opens the Properties view, or dialog box, for that node.

   If you have included a user-defined node in your message flow, and have defined one or more of its properties as configurable, you might get a warning about a custom property editor. If you define a property as a configurable property, and you have specified that it uses a custom property editor, the BAR editor cannot handle the custom property editor, and handles the property as if it is type string. This action restricts your ability change this property at deployment time.
3. Correct the error that is indicated by the message.
   For example, provide a value for a mandatory property.
4. When you have corrected all the errors, save the message flow or subflow again.

   The editor validates all the resources that you have changed, and removes the corresponding graphical indication from the nodes that you have modified successfully. Corrected errors are also removed from the Problems view.

Results

You do not have to correct every error to save your work. The editor saves your resources even if it detects errors or warnings, so that you can continue to work with them at a later date. However, you cannot deploy a resource that has a validation error. You must correct every error before you deploy a resource. Warnings do not prevent successful deployment.

Designing a message flow

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

Before you begin

Read the following concept topic: "Message flow nodes" on page 484.

About this task

When you design a message flow, consider the following questions and options:

- The mode that your integration node is working in can affect the types of message flow node that you can use and the number of message flows you can deploy. For more information, see “Restrictions that apply in each operation mode” on page 5.
- Which message flow nodes provide the function that you require. In many cases, you can choose between several nodes that provide a suitable function. You might have to consider other factors listed here to determine which message flow node is best for your overall needs. You can include built-in nodes, user-defined nodes, and subflow nodes. For more information, see “Deciding which nodes to use” on page 585.
- Whether it is appropriate to include more than one input node. For more information, see “Using more than one input node” on page 597.
- How to specify the characteristics of the input message. For more information, see “Defining input message characteristics” on page 598.
- Whether to determine the path that a message follows through the message flow, based on the content or the characteristics of the message. Several message flow nodes provide checks or examination of the message, and have output terminals that can be connected to direct certain messages to different message flow nodes. For more information, see “Using nodes for decision making” on page 1233.
- Whether it is appropriate to split message flow processing between different locations. A message flow can call another flow directly. You can deploy both flows to IBM App Connect Enterprise or IBM App Connect on IBM Cloud. You can also deploy one flow to IBM App Connect Enterprise and one flow to IBM App Connect on IBM Cloud. Callable flows also facilitate reuse because they can be called by multiple message flows. For more information, see “Splitting the processing of message flows” on page 599.
- Whether you can use subflows that provide a well-defined subset of processing. You might be able to reuse subflows that were created for another project (for example, an error processing routine), or you might create a subflow in your current project, and reuse it in several places in the same message flow. For more information, see “Subflows” on page 488.
- What response times your applications expect from the message flow. This factor is influenced by several aspects of how you configure your nodes and the message flow. For more information, see “Optimizing message flow response times” on page 2765.
- Whether you can use the destination list in the local environment that is associated with the message to determine the processing in the message flow (for example, by using RouteToLabel and Label nodes), or the target for the output messages (for example, by setting the `Destination Mode` property of the MQOutput node to Destination List). For more information, see “Creating destination lists” on page 610.
- Whether to use IBM MQ cluster queues. For more information, see “Using IBM MQ cluster queues for input and output” on page 780.
- Whether to validate input messages that are received by the input node, or output messages that are generated by the Compute node, or both. For more information, see “Validating messages” on page 610.
- Whether to view or record message structure in Trace node output. For more information, see “Viewing the logical message tree in trace output” on page 685.
- Whether your message flows access data in databases. You must configure integration nodes, databases, and database connections to enable this function, as described in “Working with databases” on page 1130. You must also configure your message flows; see “Accessing databases from message flows” on page 1134.

  If you include message flow nodes that use ESQL, for information about how to code the appropriate statements, see “Accessing databases from ESQL” on page 1135. If you want to access databases from Java nodes by using JDBC, see “Interacting with databases by using the JavaCompute node” on page 1785 or “Extending the capability of a Java message processing or output node” on page 2415.

- Whether your message flows access data in files. By using the FileInput and FileOutput nodes, your message flows can read messages from files and write messages to files in the local file system, or on a network file system that appears local to the integration node. For more information, see “Connecting client applications” on page 737.
- Whether your messages must be handled in a transaction. You can set the properties of some built-in message flow nodes to control how transactions are managed, and how messages are processed in a transaction. For more information, see “Configuring transactionality for message flows” on page 698.

  If you want to include JMSInput and JMSOutput nodes in your message flow transactions, you must consider the additional information in “Configuring JMS and SOAP nodes to support globally coordinated transactions” on page 721.
- Whether you want your messages to go through data conversion. For information about the available options, see “Configuring message flows for data conversion” on page 727.
- Whether you want to use the MQGet node. For more information about how messages are processed by the MQGet node, and a description of request-reply scenarios that uses this node, see “Working with IBM MQ” on page 753.
- How your message flows can benefit from user exits. For more information, see “Exploiting user exits” on page 2329.
- What steps to take to ensure that messages are not lost. For more information, see “Ensuring that messages are not lost” on page 786.
- How errors are handled in the message flow. You can use the facilities provided by the integration node to handle any errors that arise during message flow execution (for example, if the input node fails to retrieve an input message, or if writing to a database results in an error). However, you might prefer to design your message flow to handle errors in a specific way. For more information, see “Handling errors in message flows” on page 1883.

Deciding which nodes to use

IBM App Connect Enterprise includes many message processing nodes that you can use in your message flows.

Before you begin

Read the concept topic, “Message flow nodes” on page 484.

About this task

IBM App Connect Enterprise also provides an interface that you can use to define your own nodes, which are known as user-defined nodes.

The mode in which IBM App Connect Enterprise is running can affect the types of node that you can use; see “Restrictions that apply in each operation mode” on page 5.

Your decision about which nodes to use depends on the processing that you want to perform on your messages.

Input, output, and request nodes

Input and output nodes define points in the message flow to which client applications send messages (input nodes, such as MQInput), and from which client applications receive messages (output nodes, such as MQOutput). Client applications interact with these nodes by putting messages to, or getting messages from, the I/O resource that is specified by the node as the source or target of the messages. Although a message flow must include at least one input node, it does not have to include an output or request node.

An input node is different from other nodes because it controls when the rest of the message flow is triggered to do its processing. The input node is designed to check when there is data for the message flow to process, read that data from the transport or server, and present that data to the rest of the flow for processing. The other nodes do processing, but do not control when the flow gets invoked.

You can also use reply, request, and response nodes to interact with other applications from within a message flow; these types of node are supplied for a subset of protocols only.

- If you are creating a message flow for deployment to an integration node, you must include at least one input node to receive messages. The input node that you select depends on the source of the input messages, and where in the flow you want to receive the messages.
- If you want to send the messages that are produced by the message flow to a target application, you can include one or more output nodes. The output node that you select depends on the transport across which the target application expects to receive those messages.
- If you want to make a request, in the middle of your flow, to an external system, and put the result into the message tree, use a request node.

Nodes for manipulating, enhancing, and transforming messages

Most enterprises have applications that have been developed over many years, on different systems, using different programming languages, and different methods of communication. IBM App Connect Enterprise removes the need for applications to understand these differences by providing the ability to configure message flows that transform messages from one format to another.

For example, personal names are held in many forms in different applications. Surname first or last, with or without middle initials, uppercase or lowercase, are just some of the permutations. Because you can configure your message flow to know the requirements of each application, each message can be transformed to the correct format without modifying the sending or receiving application.

You can work with the content of the message to update it in several ways. Your choices here might depend on whether the message flow must handle predefined (modeled) messages, self-defining messages (for example, XML), or both.

A message flow can completely rebuild a message, convert it from one format to another (for example, changing order of fields, byte order, or language), remove content from the message, or introduce specific data into it. For example, a node can interact with a database to retrieve additional information, or to store a copy of the message (whole or part) in the database for offline processing.

The following examples show the importance of message transformation:

- An order entry application has a part ID in the body of the message, but its associated stock application expects it in the message header. The message is directed to a message flow that knows the two different formats, and can therefore reformat the information as it is needed.
- A data-entry application creates messages containing stock trade information. Some applications that receive this message need the information as provided, but others need additional information added to the message about the price to earnings (PE) ratio. The stock trade messages are directed to a message flow that passes the message unchanged to some output nodes, but calculates and adds the extra information for the others. The message flow does this calculation by looking up the current stock price in a database, and uses this value and the trade information in the original message to calculate the PE value before passing on the updated message.

You can also create message flows that use these nodes to interact with each other. Although the default operation of one message flow does not influence the operation of another message flow, you can force this action by configuring your message flows to store and retrieve information in an external source, such as a database.

These nodes are supplied to transform messages.

Nodes for making decisions

You can use nodes that determine the order and flow of control in the message flow in various ways to decide how messages are processed by the flow. You can also use nodes (TimeoutControl and TimeoutNotification) that determine the time, and frequency of occurrence, of events in the message flow. Routing is independent of message transformation, although the route that a message takes might determine exactly what transformation is performed on it.

For example, a money transfer application always sends messages to one other application. You might decide that every message with a transfer value of more than $10,000 must also be sent to a second application, to enable all high-value transactions to be recorded.

In another example, a national auto club offers a premier service to specific members for orders above a threshold value. Most orders are routed through the typical channels, but, if the membership number and order value meet certain criteria, the order gets special treatment.

You can also establish a more dynamic routing option by building additional routing information into the message when it is processed. Optional sets of rules are set up to receive messages according to values (destinations) set into the message. You can establish these rules such that a message is processed by one or more of the optional sets of rules, in an order that is determined by the added message content.

These nodes are provided to decide about the route that a message follows through the message flow.

Nodes for controlling time-sensitive operations
You might want a batch application process to run every day at a specific time, or you might want information to be processed and published at fixed intervals (for example, currency exchange rates are calculated and sent to banks), or you might want to take a specified recovery action if certain transactions are not completed within a defined time. For all these cases, two timeout nodes (TimeoutControl and TimeoutNotification) are provided; see “Nodes for controlling time-sensitive operations” on page 595.

Miscellaneous nodes
Other nodes exist to do the following tasks:

- Collate requests
- Create message collections
- Control the sequence of messages
- Handle and report errors
- Invoke the message flow security manager

See “Miscellaneous nodes” on page 596 for details.

Input nodes
You must include at least one input node in your message flow.

An input node is different from other nodes because it controls when the rest of the message flow is triggered to do its processing. The input node is designed to check when there is data for the message flow to process, read that data from the transport or server, and present that data to the rest of the flow for processing. The other nodes do processing, but do not control when the flow gets invoked.

CallableInput node
Use the CallableInput node in a callable flow so that you can split message flow processing between different locations. The CallableFlowInvoke node in a calling flow calls the CallableInput node of a callable flow.

DatabaseInput node
Use the DatabaseInput node to respond to events in a database. For example, the integration node can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

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

FileInput node
Use a FileInput node if the messages are contents of files.

FTEInput node
Use the FTEInput node to receive files using IBM MQ File Transfer Edition.

HTTP input node
Use an HTTPInput node if the messages are sent by a web services client.

Input node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Input node to receive messages into the subflow.

An instance of the Input node represents an In terminal. For example, if you have included one instance of the Input node, the subflow icon shows one In terminal, which you can connect to other nodes in the main flow in the same way that you connect any other node.

To deploy a message flow, it must have at least one input node. If your message flow does not contain an input node, you are prevented from adding it to the BAR file. The input node can be in the main flow, or in a message flow that is embedded in the main flow.

You can use more than one input node in a message flow. For more information, see "Using more than one input node" on page 597.

JMSInput node
Use a JMSInput node if the messages are sent by a JMS application.

KafkaConsumer node
Use the KafkaConsumer node to connect to the Kafka messaging system and to receive messages that are published on a Kafka topic.

MQInput node
Use an MQInput node if the messages arrive at the integration node on an IBM MQ queue, and the node is to be at the start of a message flow.

MQTTSubscribe node
Use the MQTTSubscribe node to receive messages that are published by applications and devices to a topic that is hosted on an MQ Telemetry Transport (MQTT) server.

.NETInput node
Use the .NETInput node to create inputs for the .NETCompute node using C#, F#, and VB templates.

SOAP input node
Use the SOAPInput node to process client SOAP messages and to configure the message flow to behave like a SOAP web services provider.

TCPIPClientInput or TCPIPServerInput node
Use a TCPIPClientInput node or a TCPIPServerInput node to create a TCP/IP connection when messages are sent through raw TCP/IP sockets.

TCPIPClientReceive or TCPIPServerReceive node
Use a TCPIPClientReceive node or a TCPIPServerReceive node to read the messages that arrive in the message flow through a TCP/IP connection.

User-defined input node
Use a user-defined input node if the message source is a client or application that uses a different protocol or transport.

WebSphere Adapters nodes
Use the WebSphere Adapters nodes to interact with Enterprise Information Systems (EIS) such as SAP, Siebel, and PeopleSoft. The following input nodes are available:

- SAPInput node
- SiebelInput node
- PeopleSoftInput node
- JDEdwardsInput

The WebSphere Adapters input nodes monitor an EIS for a particular event. When that event occurs, business objects are sent to the input node. The node constructs a tree representation of the business objects and propagates it to the Out terminal so that the data can be used by the rest of the message flow.

The WebSphere Adapters request nodes can send and receive business data. They request information from an EIS and propagate the data to the rest of the message flow.

Output nodes

If you want to send the messages that are produced by the message flow to a target application, include one or more output nodes in the flow.

CallableReply node
Use a CallableReply in a callable flow to return a response to the CallableFlowInvoke node in the message flow that called the callable flow.

EmailOutput node
Use the EmailOutput node to send an email message to one or more recipients.

FileOutput node
Use a FileOutput node if a file is the target of the output messages.

FTEOutput node
Use the FTEOutput node to write messages to files using the IBM MQ File Transfer Edition.

HTTPReply node
Use an HTTPReply node if the messages are in response to a web services client request.

JMSOutput and JMSReply nodes
Use a JMSOutput node if the messages are for a JMS destination.

The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and when you have no other routing requirements.

KafkaProducer node
Use the KafkaProducer node to connect to the Apache Kafka messaging system, and to publish messages from a message flow to a topic on a Kafka server.

MQOutput and MQReply nodes
Use an MQOutput node if the target application expects to receive messages on an IBM MQ queue, or on the IBM MQ reply-to queue that is specified in the input message MQMD.

Use an MQReply node if the target application expects to receive messages on the IBM MQ reply-to queue that is specified in the input message MQMD.

MQTTPublish node
Use the MQTTPublish node to publish an output message from a message flow to a topic that you specify on an MQ Telemetry Transport (MQTT) server.

Output node
If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a stand-alone message flow, you must include at least one Output node to propagate messages to subsequent nodes that you connect to the subflow.

An instance of the Output node represents an Out terminal. For example, if you have included two instances of the Output node, the subflow icon shows two Out terminals, which you can connect to other nodes in the main flow in the same way that you connect any other node.

Publication node
Use a Publication node to distribute the messages using the publish/subscribe network for applications that subscribe to the integration node across all supported protocols. A Publication node is an output node that uses output destinations that are identified by subscribers whose subscriptions match the characteristics of the current message.

SAPReply node
Use this node with the SAPInput node to respond to an incoming SAP event.

SOAPReply node
Use a SOAPReply node if the target application expects to receive SOAP messages in response to a message sent to the SOAPInput node.

SCAReply node
Use the SCAReply node to send a message from the integration node to the originating client in response to a message received by a SCAInput node.

TCPIPClientOutput or TCPIPServerOutput node
Use a TCPIPClientOutput node or a TCPIPServerOutput node if the messages are to be sent to the target application through raw TCP/IP sockets.

User-defined output node
Use a user-defined output node if the target is a client or application that uses a different protocol or transport.

Request nodes

If you want to make a request, in the middle of your flow, to an external system, and put the result into the message tree, use a request node.

AppConnectRESTRequest node
Use the AppConnectRESTRequest node to interact with an IBM App Connect REST API. The AppConnectRESTRequest node uses an imported Swagger document (in either JSON or YAML format), which defines the REST API and the operations that you can invoke.

CallableFlowInvoke node
Use the CallableFlowInvoke node to call a callable flow, so that you can split message flow processing between different locations. The CallableFlowInvoke node calls the CallableInput node of a callable flow. A CallableReply node in the callable flow sends a response to the CallableFlowInvoke of the calling flow.

CICSRequest node
Use the CICSRequest node to call an external CICS Transaction Server for z/OS application over TCP/IP-based IP InterCommunications (IPIC) protocol. You can create a message flow that contains a CICSRequest node, which calls an application on CICS. By using the CICS support that is provided in IBM App Connect Enterprise you can deploy CICS applications into a service-oriented architecture (SOA).

CORBARequest node
Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP). You can create a message flow that contains a CORBARequest node, which calls a CORBA server. The message flow uses an IDL file to call methods on a remote CORBA object. You can then give existing CORBA applications a new external interface; for example, a SOAP interface.

Database node
Use the Database node to interact with a database that is identified by the node properties. The Database node handles both predefined and self-defining messages. Use the ESQL editor to code ESQL functions to update database content from the message, insert new information into the database, and delete information from the database, using information in the message. Do not use the ESQL code that you develop for use in a Database node in any other type of node.

This node provides a flexible interface with a wide range of functions. It also has properties that you can use to control the way in which the interaction participates in transactions.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node properties. Use the **mqsisetdbparms** command to initialize and maintain these values.

You can update only databases from this node; you cannot update message content. If you want to update message content, use the Compute or Mapping node.

DatabaseRetrieve node
Use the DatabaseRetrieve node to ensure that information in a message is up to date. Use the node to modify a message using information from a database. For example, you can add information to a message using a key, such as an account number, that is contained in a message. Use the DatabaseRetrieve node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

FileRead node
Use the FileRead node to read a file from the middle of a message flow. The node can:

- Read the entire contents of the file.
- Read a single record.
- Rename or delete the file without reading any data.

HTTPRequest node
Use an HTTPRequest node if your message flow interacts with a web service after it has started.

IMSRequest node
Use the IMSRequest node to send a request to run a transaction on a local or remote IBM Information Management System (IMS) system, and wait for a response. IMS Connect must be configured and running on the IMS system.

JMSReceive node
Use the JMSReceive node to consume or browse messages from a JMS queue in the middle of a message flow. The node can augment the input message with result data from the received message.

LoopBackRequest node
Use the LoopBackRequest node in a message flow to create, retrieve, update, and delete data through a LoopBack connector such as MongoDB, Cloudant, or PostgreSQL.

MQGet node
Use an MQGet node to retrieve a message from an IBM MQ queue, if you want to get the message later in the message flow.

SalesforceRequest node
Use the SalesforceRequest node to make synchronous requests to Salesforce.com, to create, retrieve, update, and delete Salesforce records.

SOAP nodes
Use the SOAP nodes to process client SOAP messages and to configure the message flow to behave like a SOAP web services provider:

- SOAPRequest
- SOAPAsyncRequest
- SOAPAsyncResponse

WebSphere Adapters nodes
Use the WebSphere Adapters nodes to interact with Enterprise Information Systems (EIS) such as SAP, Siebel, and PeopleSoft. The following request nodes are available:

- SAPRequest node
- SiebelRequest node
- PeopleSoftRequest node
- JDEdwardsRequest node

WebSphere Service Registry and Repository (WSRR) nodes
Use the WebSphere Service Registry and Repository nodes to retrieve web services information:

- Use the EndpointLookup node to retrieve service endpoint information held in the WebSphere Service Registry and Repository.
- Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository.

Nodes for manipulating, enhancing, and transforming messages
Optionally, include nodes to change messages.

Compute node
Use the Compute node to:

- Manipulate message content

- Transform the message in some way
- Interact with a database to modify the content of the message or the database and pass on one or more new messages

You can use this node to manipulate predefined and self-defining messages.

Use the ESQL editor to create an ESQL module, specific to this node, that contains the statements that define the actions to perform against the message or database. Do not use the ESQL code that you develop for use in a Compute node in any other type of node.

You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

If possible, perform your message manipulation requirements in a single Compute node. Fewer, more complex Compute nodes perform better than a larger number of simpler nodes because the integration node parses the message on entry to each Compute node.

.NETCompute node

Use the .NETCompute node to:

- Build messages.
- Interact with Microsoft .NET Framework (.NET) or Component Object Model (COM) applications.
- Transform messages from one format to another.
- Copy messages between parsers.
- Use a .NET programming language to examine an incoming message and, depending on the message content, propagate it unchanged to one of the .NETCompute node output terminals. The node behaves in a similar way to a Filter node, but uses .NET languages instead of ESQL to determine which output terminal to use.
- Use .NET to change part of an incoming message and propagate the changed message to one of the output terminals.
- Use .NET to create and build an output message that is independent of the input message.

JavaCompute node

Use the JavaCompute node to:

- Examine an incoming message and, depending on its content, propagate it unchanged to one of the node's output terminals. The node behaves in a similar way to a Filter node, but uses Java instead of ESQL to determine which output terminal to use.
- Change part of an incoming message and propagate the changed message to one of the output terminals.
- Interact with a database through a JDBC type 4 connection to modify the content of the message or the database and pass on one or more new messages
- Create and build a new output message that is independent of the input message.

Mapping node

Use the Mapping node to:

- Manipulate graphically message content
- Enrich a message with data from a database
- Interact with a database to modify the content of the message or the database and pass on one or more new messages

You can use this node to manipulate predefined and self-defining messages.

Use the Graphical Data Mapping editor to create a message map, specific to this node, that contain the data transforms that define the actions to perform against the message or database.

XSLTransform node

Use the XSLTransform node (formerly known as the XMLTransformation node) to transform an input XML message into another format using XSLT style sheets and to set the message domain, message set, message type, and message format for the generated message. It is imperative that the data can be parsed into an XML message. The style sheet, using the rules that are defined in it, can perform the following actions:

- Sort the data
- Select data elements to include or exclude based on some criteria
- Transform the data into another format

The Xalan-Java transformation engine (Apache Xalan-java XSLT processor) is used as the underlying transformation engine. For more information about XML Transformations, the W3C specification of the syntax, and semantics of the XSL Transformations language for transforming XML documents into other XML documents, see W3C XSL Transformations.

You can deploy style sheets and XML files to integration node integration servers, to help with style sheet and XML file maintenance.

JMSMQTransform node

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the IBM MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

MQJMSTransform node

Use the MQJMSTransform node to receive messages that have an IBM MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

SOAPEnvelope and SOAPExtract nodes

Use the SOAPEnvelope and SOAPExtract nodes to add or remove SOAP envelopes from the SOAP message body. You can use the SOAPExtract node both to extract the envelope, and to route the message based on the message content to a Label node.

Header nodes

Use the HTTPHeader, JMSHeader, or MQHeader nodes to manipulate HTTP, JMS, and IBM MQ transport headers and their properties without writing compute nodes. You cannot use these nodes to change the message body.

- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPInput, HTTPResponse, HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.
- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.

User-defined processing node

Use a user-defined node processing node to handle specific requirements that are not met by the built-in nodes.

For example, if your node accesses a database, include a user-defined node to interact with the database. You can control the way in which the database is accessed by this node by specifying user and password information for the data source that you specify in the node property. Use the **mqsisetdbparms** command to initialize and maintain these values.

Cloned node

Use a Cloned node to create reusable templates from .NETInput nodes that you have created and customized in your message flow.

Cloned nodes have no special properties, and you cannot select a new Cloned node from a palette drawer. You must create it from an existing node in your message flow, and then save it into the palette drawer. For more information about using the Cloned node type, see "Cloning a .NETInput node" on page 1810.

When saved in a palette drawer, you can create an instance of your saved Cloned node by dragging into your message flow. You can therefore use a Cloned node to be able to reproduce your customized parameters and values without manually configuring a new node instance each time. You can save multiple templates in the palette, if each Cloned node you create has a unique name.

Nodes for making decisions

Optionally, use nodes that determine the order and flow of control in the message flow to decide how messages are processed by the flow.

Nodes for making decisions

Validate node

Use the Validate node to check that the message that arrives on its input terminal is as expected. You can check that the message has the expected message template properties (message domain, message set, and message type) and that the content of the message is correct. You can check the message against one or more of the message domain, message set, or message type values.

The Validate node replaces the Check node, which is deprecated. The Validate node works in the same way as the Check node, but it has more Validation properties to enable the validation of message content by parsers that support that capability.

Filter node

Use the Filter node with an ESQL statement to determine the next node to which the message is sent by this node. Do not use the ESQL code that you develop for use in a Filter node in any other type of node.

The node terminals are True, False, Unknown, and Failure. The message is propagated to the True terminal if the test succeeds, and to the False terminal if it fails. If the statement cannot be resolved (for example, it tests the value of a field that is not in the input message), the message is propagated to the Unknown terminal. If any other error is detected, the message is propagated to the Failure terminal.

The test in the ESQL statement can depend on message content, database content, or a combination of the two.

If you refer to a database, you can control how it is accessed by this node by specifying user and password information for each data source that is defined in the registry on the integration node system. Use the **mqsisetdbparms** command to initialize and maintain these values.

Use this node in preference to the Compute node to provide message selection and routing; the Filter node is more efficient for this task.

FlowOrder node

You can connect the terminals of this node to force the message to be processed by one sequence of nodes, followed by a second sequence of nodes.

Passthrough node

Use the Passthrough node to enable version control of a subflow at run time. Use this node to add a label to your subflow. By combining this label with a reserved word replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you include the correct version keywords in the label, you can see the value of the label:

- Stored in the BAR file, by using the **mqsireadbar** command

- As last deployed to a particular integration node, on the properties of a deployed message flow in the IBM App Connect Enterprise Toolkit
- In the integration node, if you enable user trace for that message flow

Route node
Use the Route node to direct messages that meet certain criteria down different paths of a message flow. For example, you can forward a message to different service providers based on the request details. You can also use the Route node to bypass unnecessary steps. For example, you can check to see whether certain data is in a message, and run a database lookup operation only if the data is missing. If you set the `Distribution Mode` property to `All`, you can trigger multiple events that each require different conditions. For example, you can log requests that relate to a particular account identifier, and send requests that relate to a particular product to be audited.

Use the Route node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

RouteToLabel node
Use the RouteToLabel node after a Compute node or a JavaCompute node for complex routing. Define a list of destinations in a Compute or JavaCompute node that are acted on by the RouteToLabel node. The RouteToLabel node interrogates the destinations and passes the message on to the corresponding Label node.

DatabaseRoute node
Use the DatabaseRoute node to route a message by using information from a database with applied XPath routing expressions. The node looks up a collection of named column values from a located database row and synchronously applies one or more XPath expressions to these acquired values. Use the DatabaseRoute node to implement message routing with minimal programming logic. For more advanced routing scenarios, use a Compute node or a JavaCompute node.

Label node
Use the Label node as a target for the next sequence of one or more nodes that are to process a message. Use this node in combination with the RouteToLabel node for all types of messages, or with the SOAPExtract node for SOAP messages.

The Label node routes the message to the next node in the flow and completes no processing.

ResetContentDescriptor node
Use the ResetContentDescriptor node to set new message properties that are used when the message bit stream is next parsed by a subsequent node in the message flow.

Mapping node
Use the Mapping node to route a message.

Nodes for controlling time-sensitive operations
Run processes at specific times, or at fixed intervals, and take action if transactions are not completed within a defined time.

TimeoutControl node
Use a TimeoutControl node and a TimeoutNotification node together in a message flow to control events that occur at a specific time or at defined time intervals. The TimeoutControl node receives an input message that contains a timeout request. All or part of this input message is validated and stored to be propagated by an associated TimeoutNotification node in the message flow. The input message is also propagated unchanged to the next node in the message flow.

More than one TimeoutControl node can be associated with each TimeoutNotification node.

TimeoutNotification node
Use a stand-alone TimeoutNotification node to generate messages that are propagated at configured times or time intervals to the next node in the message flow for further processing.

Miscellaneous nodes

Nodes to perform a variety of tasks.

Nodes for collating requests

Group nodes and aggregation nodes provide alternative ways to collate related requests and responses. Use these nodes to generate several requests in response to one input message, to control and coordinate the responses that are received in response to those requests, and to combine the information that is provided by the responses to continue processing.

- Group nodes: GroupScatter, GroupGather, and GroupComplete.
- Aggregate nodes: AggregateControl, AggregateReply, and AggregateRequest

For more information about these nodes, see "Group nodes and aggregation nodes" on page 596.

Node for creating message collections

Use the Collector node to generate collections of messages and make multiple synchronous or asynchronous requests in parallel. The Collector node does not need an initial fan-out stage, and can group unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection has completed.

Nodes for controlling the sequence of messages

Use the Sequence node to apply a sequence number to a message received from an input source, and the Resequence node to change the sequence of messages that contain a sequence number. You can divide messages into sequence groups and use the Sequence and Resequence nodes to control the sequences in groups of messages that arrive at the nodes.

Nodes for handling and reporting errors

Use the following nodes to affect error handling and reporting:

Trace node
Include a Trace node to generate one or more trace entries to record what is happening in the message flow at this point.

TryCatch node
Include a TryCatch node to control the error processing when exceptions are thrown.

Throw node
Include a Throw node to force an exception to be thrown, and specify the identity of the exception, to make it easier to diagnose the problem.

Node for invoking the message flow security manager

Use the SecurityPEP node to invoke the message flow security manager at any point in the message flow between an input node and an output (or request) node. The SecurityPEP node enables you to invoke the security manager even if your input nodes do not support message flow security. You can also use the SecurityPEP node to invoke different aspects of security (for example, authentication and authorization) at different points in the message flow.

Group nodes and aggregation nodes

Group nodes and aggregation nodes provide alternative ways to collate related requests and responses. Use these nodes to generate several requests in response to one input message, to control and coordinate the responses that are received in response to those requests, and to combine the information that is provided by the responses to continue processing.

The Group nodes provide the ability to create fan-out/fan-in style static and dynamic aggregation scenarios that are stateless and highly performant without requiring an IBM MQ queue manager to be available for the integration server. The node is responsible for creating a new group, and enables downstream nodes to send requests that are marked as part of the group. (The list of currently supported downstream nodes for a group is: MQOutput node (the node must have the "Request" attribute set), HTTPAsyncRequest node, CallableFlowAsyncInvoke node, and RESTAsyncRequest node.)

The Aggregation nodes provide a similar capability, but make use of storage queues that are controlled by IBM MQ so you must install IBM MQ on the same computer as your integration server if you want to use the capabilities that are provided by the aggregation nodes. For more information about the queues that are required by aggregation nodes, see “Configuring the storage of events for aggregation nodes ” on page 229.

Note: The group nodes are not a drop-in replacement for the aggregate nodes, therefore customers who wish to migrate from using the aggregate nodes to the group nodes will need to redesign their flows.

There is no group node that is equivalent to the AggregateRequest node, rather supported output nodes now directly specify whether they are participating in a group or not by their node properties. Custom transports can still be enabled using Environment overrides.

Testing has shown that the group nodes provide better overall performance when compared to the aggregate nodes, with reduced message processing times and CPU costs.

You should use the group nodes if you have the following requirements:

- You want a stateless integration server
- You want highly performant aggregations that scales to high numbers of branches
- You do not have access to, or do not want to use, IBM MQ
- You will deploy all flows responsible for orchestrating the aggregations to a single integration server (note this does not include the backend services)

You should use the Aggregate nodes if you have the following requirements:

- You require that all parts of the aggregation, including control data, received replies and sent requests, are recoverable if the integration server shuts down
- You want to deploy the flows responsible for orchestrating the aggregations across multiple integration servers within the same integration node
- Your backend replies are large, which may cause excessive memory consumption if not stored on disk while waiting for the aggregation to complete

Using more than one input node

You can include more than one input node in a single message flow.

Before you begin

Read the following concept topic:

- “Message flow nodes” on page 484

About this task

You might find this useful in the following situations:

- The message flow provides common processing for messages that are received from multiple transports. For example, a single message flow might handle:
  - Data in messages received from IBM MQ, and therefore through an IBM MQ queue and an MQInput node
  - Messages that are received from native IP connections (a Real-timeInput node)

- You need to set standard properties on the MQInput node if input messages:
  - are predefined, and
  - are all received from IBM MQ, and
  - do not include an MQRFH2 header.

  If the required standard properties are not always the same for every message, you can include more than one input node and configure each to handle a particular set of properties.

  This requirement is not necessary for self-defining messages.
- Each input node in a message flow causes the integration node to start a separate thread of execution. Including more than one input node might improve the message flow performance. However, if you include multiple input nodes that access the same input source (for example, an IBM MQ queue), the order in which the messages are processed cannot be guaranteed. If you want the message flow to process messages in the order in which they are received, this option is not appropriate.

  If you are not concerned about message order, consider using additional instances of the same message flow rather than multiple input nodes. If you set the `Additional Instances` property of the message flow when you deploy it to the integration node, multiple copies of the message flow are started in the integration server. This is the most efficient way of handling multiple instances.

Defining input message characteristics

When a message is received by an input node in a message flow, the node detects how to interpret that message by determining the domain in which the message is defined and starting the appropriate parser.

Before you begin

Read the following concept topic:

- "Parsers" on page 520

About this task

You can provide message domain information to the input node in one of two ways:

- You can configure the built-in input nodes to indicate the message domain, and therefore the parser to be started, for each message that is received.
- You can set values in the input message itself that specify this information. Include an MQRFH2 header, which contains a folder that defines the message characteristics. This approach is more flexible because it means that the input node can start the appropriate parser based on the content of each message.

If the input message is defined in the MRM domain, and is therefore interpreted by the MRM parser, you must specify the following additional properties:

- The `Message model`, which is the message model in which the message is defined
- The `Message type`, which is defined by the message model
- The `Message format`, which defines the physical characteristics of the message

The way that these properties are set depends upon the type of message, or node, that you want to use:

- If the message is an IBM MQ message, these properties can be set either in the input node or in the MQRFH2 header of the incoming message. If the properties are set in both, the properties of the MQRFH2 header take precedence. If the properties are not found in either the node or the MQRFH2 header, the default value is empty and the BLOB parser is used.
- If the message is a JMS message, the property that is set on the node takes precedence. If the `Message domain` is empty, the `Message domain` is, by default, derived according to certain criteria following a predetermined order of precedence; see "JMS message payload and appropriate parser" on page 863.

- If the input message belongs to a `Message domain` other than those for which a parser is supplied, you must provide a user-defined parser to handle it, and a user-defined input node to accept it for processing in the message flow. Check the documentation provided with the user-defined parser and node for further information.
- If the `Message domain` is in a TimeoutControl node, an empty `Message domain` has either of the following results:
  - If the `Stored message location` property is also empty, the full message is stored. When the message comes back at TimeoutNotification, it is parsed in the same way as the original message.
  - If the `Stored message location` property is not empty, a partial message is stored and no parser is associated, therefore, by default, it is treated as BLOB.
- If the `Message domain` is in a ResetContentDescriptor node, an empty `Message domain` has either of the following results:
  - If `Reset message domain` is cleared, the domain is not reset.
  - If `Reset message domain` is selected, the default is BLOB.
- If the input node cannot determine the message characteristics, the default value is empty and the message is considered to be in the BLOB domain, and the BLOB parser is started.

Splitting the processing of message flows

Process parts of your message flow in different locations by creating one message flow that calls another message flow.

Before you begin

Read the concept information in "Callable message flows" on page 491.

About this task

Callable flows allow you to split your message flow processing between different locations. Callable message flows also facilitate reuse because they can be called by multiple message flows. You can split your processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, or between different locations in IBM App Connect Enterprise.

In App Connect Enterprise message flows, the calling flow uses a CallableFlowInvoke node to call a CallableInput node in the callable flow. The callable flow then uses a CallableReply node to return information to the CallableFlowInvoke node in the original flow.

Note: This topic is about using the callable flow technique between App Connect Enterprise message flows. You can also use the callable flows technique with on-cloud event-driven flows and API flows in IBM App Connect on IBM Cloud; for example, see the tutorial: Sharing data and processing from on-premises activities with on-cloud SaaS applications, by using an on-cloud callable flow.

If your initial and callable flows are in the same integration server, they can communicate with each other as soon as you deploy them. If your flows are in different integration servers, or one is in IBM App Connect Enterprise while the other is in IBM App Connect on IBM Cloud, you must configure communication between them. The flows use a Switch server to route data, and connectivity agents to connect securely to the Switch server.

You can split flows into multiple callable flows. However, for simplicity, the following tasks describe how to create two flows only: a calling flow and a callable flow.

Procedure

To split the processing of your message flows, complete the following steps.

- Create your initial and callable flows. For more information, see "Developing synchronously callable message flows" on page 600 and "Developing asynchronously callable message flows" on page 602.

- If your flows are split between different integration servers in IBM App Connect Enterprise, configure connectivity between the flows. For more information, see “Preparing the environment to split processing between different integration servers” on page 222.
- If your flows are split between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, configure connectivity between the flows. For more information, see “Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud” on page 225.

Developing synchronously callable message flows

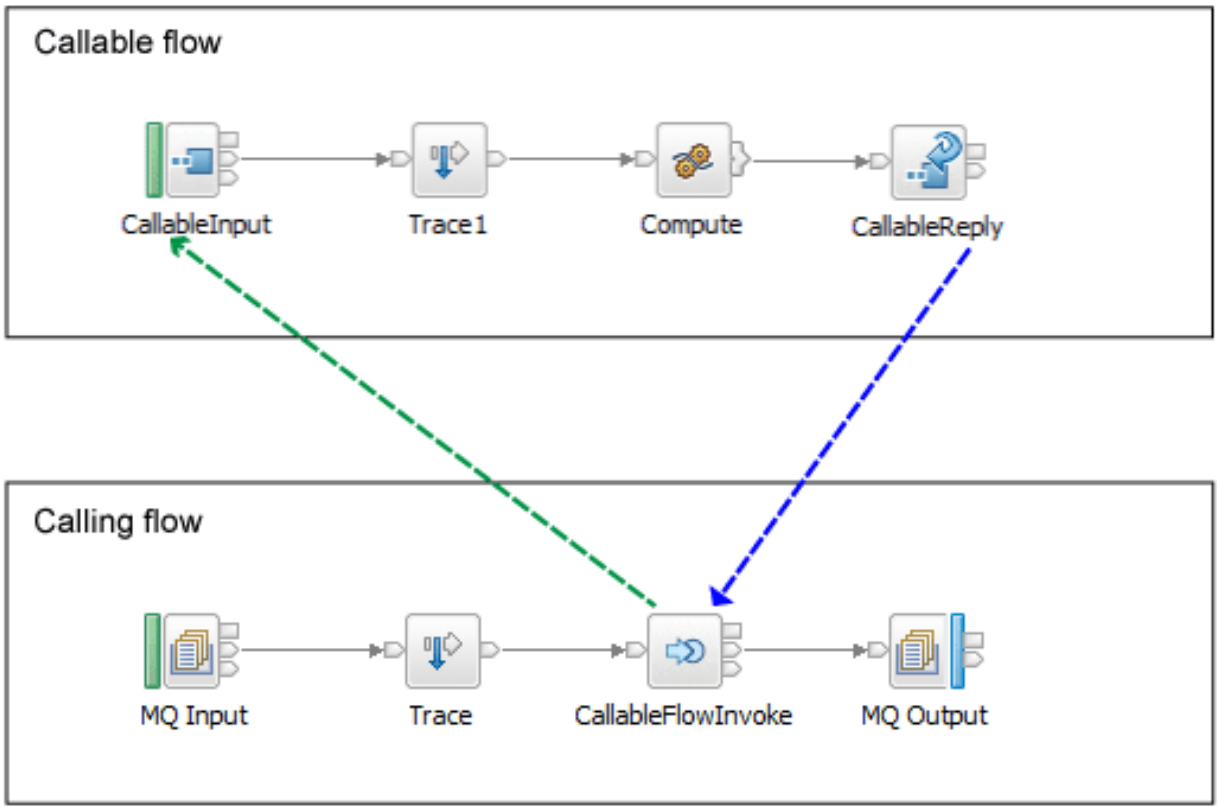
Create a calling message flow that contains a CallableFlowInvoke node. Create a callable message flow that contains CallableInput and CallableReply nodes.

Before you begin

Read the concept information in “Callable message flows” on page 491.

About this task

The following steps describe how to create a calling flow and a callable flow that can send information to each other.



For information about calling a flow asynchronously, see “Developing asynchronously callable message flows” on page 602.

Procedure

1. In the IBM App Connect Enterprise Toolkit, create the calling message flow, which must include a CallableFlowInvoke node.

   The CallableFlowInvoke node parses the incoming message in full so that it is in a suitable format to send to the CallableInput node. Therefore, you should validate the message before it reaches the CallableFlowInvoke node. If the message fails validation at this point, it can be rolled back.

2. In a different application, create a callable message flow, which must begin with a CallableInput node, and contain a CallableReply node.

   A CallableFlowInvoke node calls a callable flow by referring to the endpoint name on the CallableInput node, and the application that contains the callable flow. Therefore, you must include all callable flows in applications.
3. On the CallableInput node of the callable flow, use the `Endpoint Name` property to provide a name for the callable flow.

   Application and endpoint name pairs must be unique on a single integration server. You can have multiple callable flows that share the same application and endpoint names, but they must be in different integration servers. In this case, the Switch server acts as a load balancer.
4. On the CallableFlowInvoke node of the calling flow, set the following properties.

Table 11. CallableFlowInvoke node properties

Property	Value
Target Application	Set this property to the name of the application that contains the callable flow.
Target Endpoint Name	Enter the name of the `Endpoint Name` property of the CallableInput node. This name is case sensitive. The `Target Endpoint Name` must match the `Endpoint Name` of the CallableInput node exactly.
Request timeout (sec)	Set the time within which the callable flow must be called, in seconds. If the callable flow is not called within the specified time, an error message is issued.
Call Preference	• If your main and callable flows are in the same integration server, set this property to **`Prefer local calls`**. • If your flows are split between different integration servers, or between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, set this property to **`Remote calls only`**.

5. Package the applications that contain your message flows into BAR files.
6. Deploy the BAR files to the appropriate integration servers.

   If you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, deploy one BAR file on premises and upload the other to the cloud. For more information about deploying BAR files to IBM App Connect on IBM Cloud, see Running enterprise integration solutions in App Connect on IBM Cloud in the IBM Integration community.

Results

When you pass a message into the main message flow, the CallableFlowInvoke node sends the contents of the message body and local environment folders to the CallableInput node of the callable flow. When the callable flow completes processing, the CallableReply node returns the message body and local environment folder data to the CallableFlowInvoke node of the main flow.

What to do next

If your callable flows are in different integration servers, you need to create a Switch server (which routes data) and connectivity agents to allow the flows to communicate securely. For more information, see “Preparing the environment for callable flows” on page 222.

Developing asynchronously callable message flows

Create a calling message flow that contains a CallableFlowAsyncInvoke node. Create a callable message flow that contains CallableInput and CallableReply nodes. Create a response message flow that contains a CallableFlowAsyncResponse node.

Before you begin

Read the concept information in "Callable message flows" on page 491.

About this task

This topic explains how to split processing between asynchronously callable message flows, by using the CallableFlowAsyncInvoke and CallableFlowAsyncResponse nodes. You can also choose to share data between the flows that contain these nodes (the calling flow and the response flow) by storing and retrieving data in the `UserContext` folder in the Environment. For information about sharing data between the calling flow and the response flow, see "Sharing data between a calling flow and a response flow" on page 603.

Procedure

The following steps describe how to create a calling flow, a callable flow, and a response flow that can send information to each other.

1. In the IBM App Connect Enterprise Toolkit, create the calling message flow, which must include a CallableFlowAsyncInvoke node.

   The CallableFlowAsyncInvoke node parses the incoming message in full so that it is in a suitable format to send to the CallableInput node. Therefore, you should validate the message before it reaches the CallableFlowAsyncInvoke node. If the message fails validation at this point, it can be rolled back.
2. In a different application, create a callable message flow, which must begin with a CallableInput node, and contain a CallableReply node.

   A CallableFlowAsyncInvoke node calls a callable flow by referring to the endpoint name on the CallableInput node, and the application that contains the callable flow. Therefore, you must include all callable flows in applications.
3. On the CallableInput node of the callable flow, use the `Endpoint Name` property to provide a name for the callable flow.

   Application and endpoint name pairs must be unique on a single integration server. You can have multiple callable flows that share the same application and endpoint names, but they must be in different integration servers. In this case, the Switch server acts as a load balancer.
4. On the CallableFlowAsyncInvoke node of the calling flow, set the following properties:

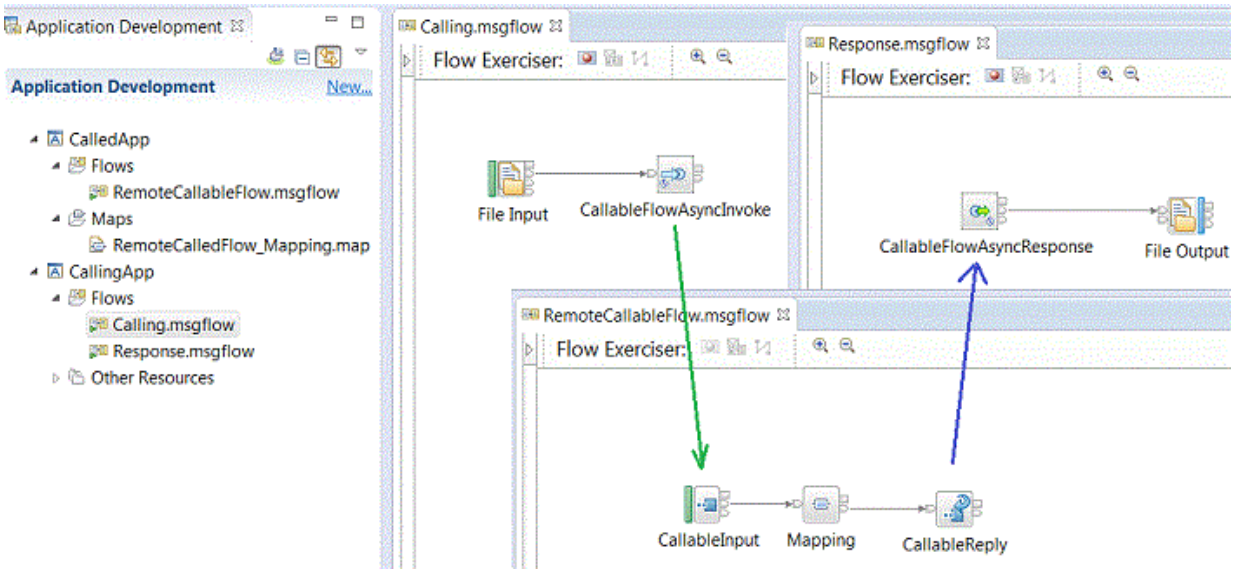
Table 12. CallableFlowAsyncInvoke node properties

Property	Value
Target Application	Set this property to the name of the application that contains the callable flow.
Target Endpoint Name	Enter the name of the `Endpoint Name` property of the CallableInput node. This name is case sensitive. The `Target Endpoint Name` must match the `Endpoint Name` of the CallableInput node exactly.
Request timeout (sec)	Set the time within which the callable flow must be called, in seconds. If the callable flow is not called within the specified time, an error message is issued.

Table 12. CallableFlowAsyncInvoke node properties (continued)	
Property	**Value**
Call Preference	• If your main and callable flows are in the same integration server, set this property to **`Prefer local calls`**. • If your flows are split between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, set this property to **`Remote calls only`**.

5. In a different application, or in the same application that was used in step 1, create a response message flow that starts with a CallableFlowAsyncResponse node.
6. Set the **`Unique identifier`** property of the CallableFlowAsyncResponse node to match the value set in the CallableFlowAsyncInvoke node that was created in step 1.

   You now have three separate message flows: a calling flow, a callable flow, and a response flow:



7. Package the applications that contain your calling and response message flows into a BAR file, and package the called message flow into a separate BAR file.
8. Deploy the BAR files to the appropriate integration servers.

   If you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, deploy one BAR file on premises and upload the other to the cloud. For more information about deploying BAR files to the cloud, see "Deploying integration solutions to a production environment" on page 2480.

Results

When you pass a message into the main message flow, the CallableFlowAsyncInvoke node sends the contents of the message body and local environment folders to the CallableInput node of the callable flow and then completes. When the callable flow completes processing, the CallableReply node sends the message body and local environment folder data to the CallableFlowAsyncResponse node in the separate response flow.

Sharing data between a calling flow and a response flow

You can share data between a flow that contains a CallableFlowAsyncInvoke node (the calling flow) and a flow that contains a CallableFlowAsyncResponse node (the response flow), by storing and retrieving data in the `UserContext` folder in the Environment.

Before you begin

- Read the concept information in "Callable message flows" on page 491.

- Create a calling flow, a callable flow, and a response flow, as described in “Developing asynchronously callable message flows” on page 602.

About this task

The user context is a storage area that is used to pass data that is not part of the message between nodes in a message flow. For example, by using the user context, you can pass state from the CallableFlowAsyncInvoke node in the calling message flow to the CallableFlowAsyncResponse node in the response message flow. However, this user context data is not accessible to the called message flow.

To store user context data, you can either set a value in the `Environment.CallableFlow.UserContext` environment variable, or create and populate child folders below it in the message tree. For example, you can use the following command to specify context data to be stored the environment:

```
SET Environment.CallableFlow.UserContext = 'myData';
```

The specified context data is shown in the `UserContext` element in the message tree:

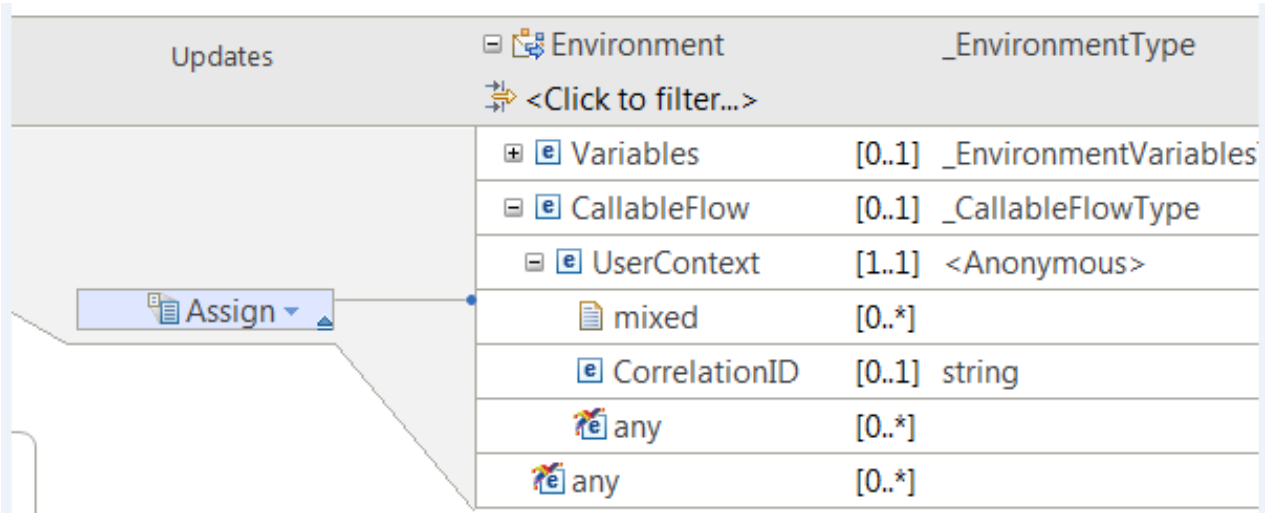
```
<environment>
  <CallableFlow>
     <CorrelationID>43414c4c0100000000000000f3264f744805000000000000</CorrelationID>
     <UserContext>myData</UserContext>
```

Alternatively, the user context can be a folder with a subtree, which is preserved and re-created:

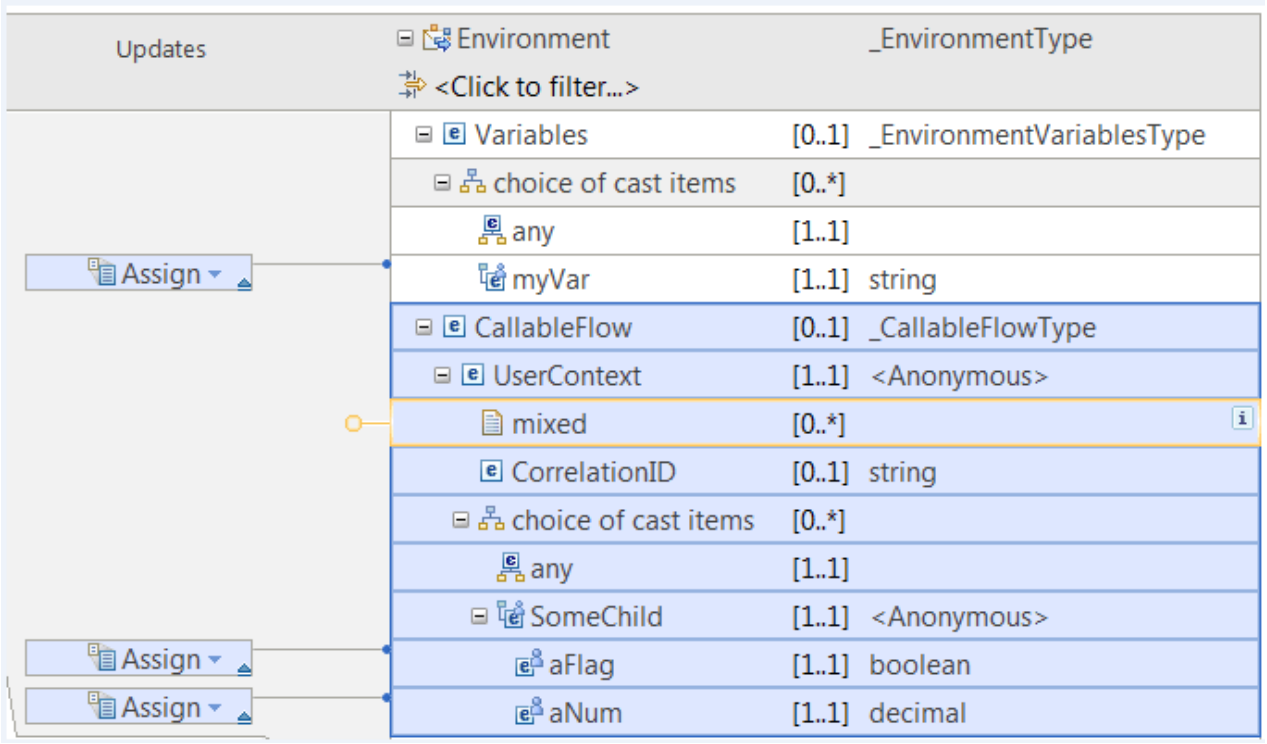
```
<environment>
    <CallableFlow>
      <CorrelationID>43414c4c0100000001000000603a527418280000000000000</CorrelationID>
      <UserContext>
        <c1>
           <c2>myData</c2>
```

When the user context data has been specified, the CallableFlowAsyncResponse node can subsequently retrieve it.

You can use a Mapping node to set a value, by mapping to the `mixed` child element `Environment.CallableFlow.UserContext.mixed`, as shown in the following diagram:



To use a mapping node to create sub-elements, use user-defined elements or cast the `xsd:any` and map to them, as shown in the following diagram:



To access the user context data after a CallableFlowAsyncResponse node, read the value or children of the `Environment.CallableFlow.UserContext` environment variable.

Preparing the environment to split processing between different integration servers

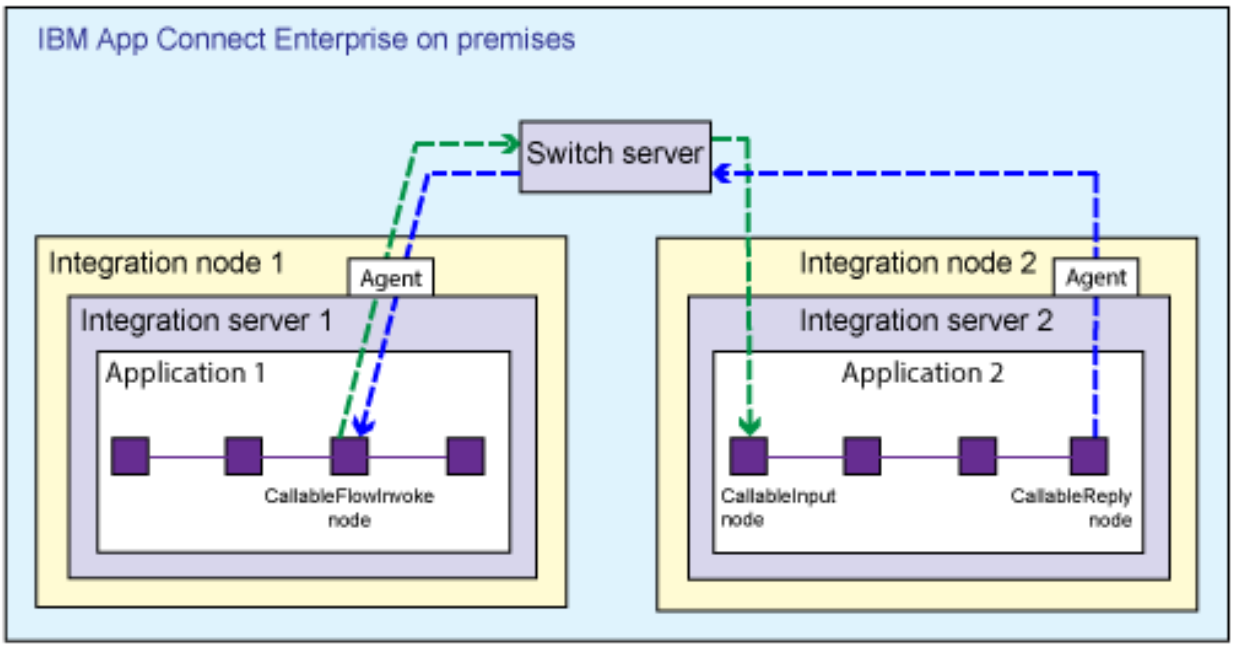
When you split message flow processing between different integration servers, you must configure a data router and a connectivity agent. The integration servers can be running on different integration nodes.

Before you begin

Read the concept information in "Callable message flows" on page 491.

About this task

When you split processing between different integration servers, your flows communicate securely by using a Switch server and connectivity agents. The Switch server is a special kind of integration server that routes data. The connectivity agents contain the certificates that your flows require to communicate securely with the Switch server. A connectivity agent must be running in each integration server where you have deployed message flows.



If your callable flows and the flows that call them are ***all*** deployed on premises, you must create and configure the Switch server on premises. You must run a command that creates some configuration files. You use one of the configuration files to create the Switch server, and the other file to configure connectivity agents for each integration server where you have deployed callable flows or flows that call them. The steps for this procedure are given in this topic.

This on-premises scenario is supported on Windows and Linux only.

If you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, the Switch server is created and managed for you in the cloud. You must download an agent configuration file from the cloud, and use it to configure the on-premises connectivity agent for each integration server. For more information about configuring connectivity agents for use with IBM App Connect on IBM Cloud, see "Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud" on page 225.

Procedure

To prepare the environment to split processing between integration servers on premises, complete the following steps.

1. Start an IBM App Connect Enterprise command environment.
2. Create the required configuration files, by running the **iibcreateswitchcfg** command.
   You need to run the command in a directory for which you have permission to write. Alternatively, you can specify an output directory. For example, the following command creates the configuration files and saves them in the `temp` directory.

   Windows On Windows:

   ```
   iibcreateswitchcfg /output c:\temp
   ```

   Linux On Linux:

   ```
   iibcreateswitchcfg -output /temp
   ```

   If this command is successful, you see the following responses:

   ```
   Generated self signed certificate file 'c:\temp\adminClient.p12'
   Generated switch configuration file 'c:\temp\switch.json'
   ```

```
Generated agentx configuration file 'c:\temp\agentx.json'
```

This command creates two JSON configuration files, and a certificate, which is reserved for future use. One file (`switch.json`) is used to create the Switch server. The other file (`agentx.json`) is used to configure the agent for each integration server where your flows are deployed.

3. Create the Switch server, by running the **iibswitch** command to use the configuration file (`switch.json`) that you created in step 2.

   Windows For example, on Windows:

   ```
   iibswitch create switch /config c:\temp\switch.json
   ```

   Linux On Linux:

   ```
   iibswitch create switch -c /temp/switch.json
   ```

   If the command is successful, you see a "Successful command completion" response. The Switch server is created in a special integration node called ACESWITCH_NODE_V11.

4. Optional: To test that the Switch server is created and running, run the following command: **mqsilist** ACESWITCH_NODE_V11.

   If the Switch server is running, you see the following response:

   ```
   BIP1286I: Integration server 'IIBSWITCH_SERVER' on integration node 'ACESWITCH_NODE_V11' is
    running.
   ```

5. Configure a connectivity agent for each integration server where you have deployed callable flows or flows that call them.

   You can configure the agent by copying the configuration file, `agentx.json` (that you created in step 2), to the server's `iibswitch/agentx` directory.

   - For an independent integration server: *work directory*`/config/iibswitch/agentx`
   - For an integration server under an integration node: *$MQSI_WORKPATH*`/config/`*Node_name*`/`*Server_name*`/iibswitch/agentx`

   For each affected integration server, the agent should automatically establish a connection to the Switch server. For example, in the App Connect Enterprise console for an independent integration server you should see the following messages:

   ```
   The connection agent for remote callable flows has established a connection to the Switch
    server with URL 'wss://localhost:9011'.
   The integration server component 'agentx' has been started.
   ```

   Similarly, in the Windows Application log, you should see the following messages for an integration server (N101ISAA) under an integration node (NODEIPL01):

   ```
   ( NODEIPL01.N101ISAA ) The connection agent for remote callable flows has established a
    connection to the Switch server with URL ''wss://localhost:9011''.

   A connection agent for remote callable flows has been configured on this integration server.
    The connection agent has successfully connected to the Switch server.

   No response required.
   ----------------------------------------------------------------------------------------------
   ( NODEIPL01.N101ISAA ) The integration server component ''agentx'' has been started.

   The specified component has been started and is available for use.

   No response required.
   ```

Results

A message flow that is deployed to one configured integration server can now communicate securely with a message flow on another configured integration server. For more information about developing these message flows, see “Developing synchronously callable message flows” on page 600.

Preparing the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud

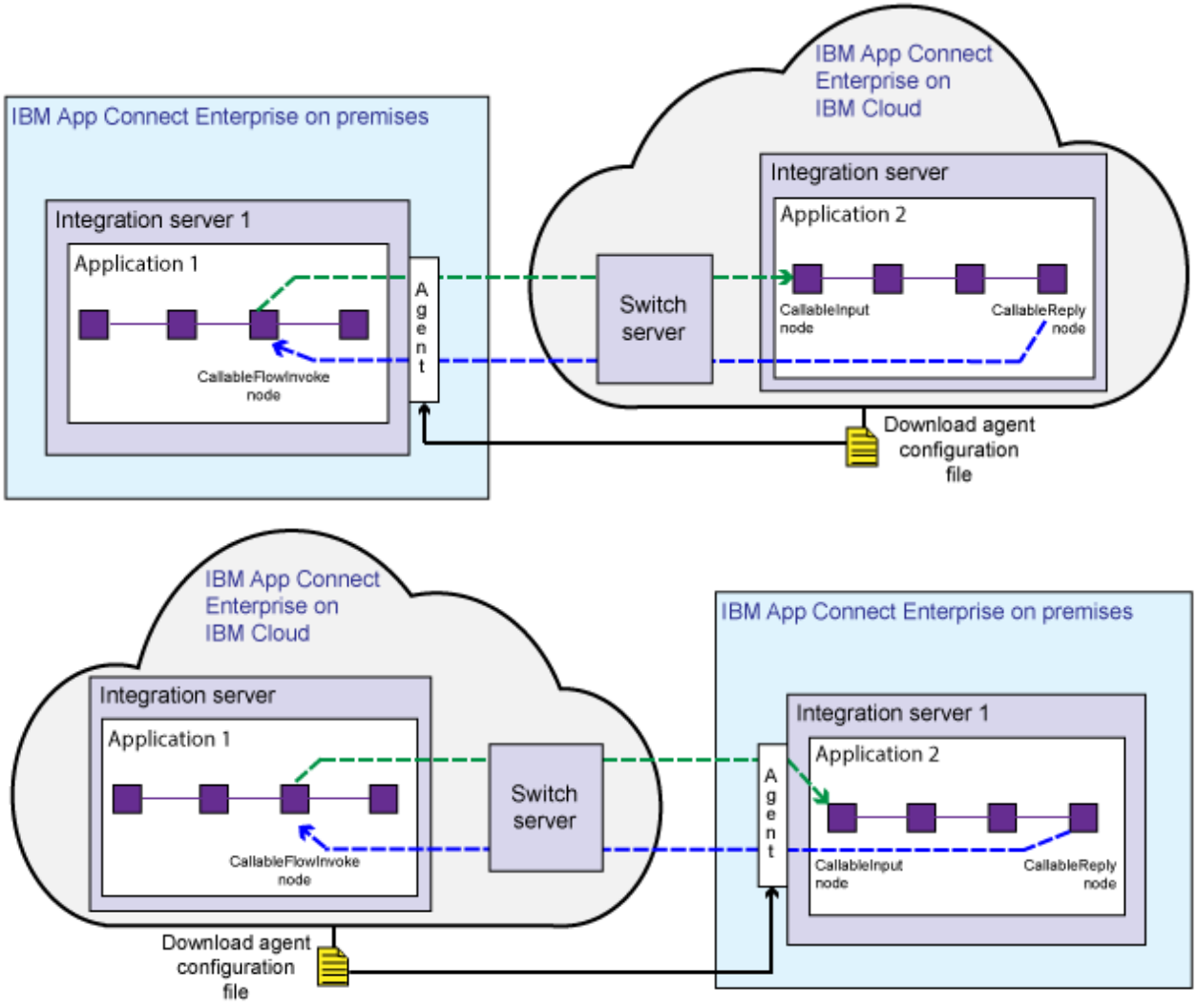
When you split message flow processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, you must configure a connectivity agent.

Before you begin

These instructions assume that you have IBM App Connect Enterprise installed on premises, and that you have created an App Connect service in IBM Cloud.

About this task

When you split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, your flows communicate by using a Switch server and connectivity agent. The Switch server, which routes data, is managed for you by IBM App Connect on IBM Cloud. The connectivity agent contains the certificates that your flows require to communicate securely with the Switch server. A connectivity agent must be running in the IBM App Connect Enterprise integration server where you have deployed your on-premises message flows.



The Switch server is already created in IBM App Connect on IBM Cloud. You must download an agent configuration file from the cloud, and use it to configure the on-premises connectivity agent. Your callable flows use this agent to communicate with each other securely, through the Switch server.

Procedure

To prepare the environment to split processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, complete the following steps.

1. In IBM App Connect on IBM Cloud, click **Callable flows**  .

   If you have not configured callable flows before, the list will be empty.
2. Click **Connect callable flows**.

   A dialog box opens to set up your agent.
3. Click **Download the configuration** and save the agent configuration file (`agentx.json`) to convenient location, and then use it to configure the on-premises integration servers.

   Copying the `agentx.json` file, to the on-premises server's `iibswitch/agentx` directory.

   - For an independent integration server: *work directory*`/config/iibswitch/agentx`
   - For an integration server under an integration node: *$MQSI_WORKPATH*`/config/`*Node_name*`/`*Server_name*`/iibswitch/agentx`

   For each affected integration server, the agent should automatically establish a connection to the Switch server. For example, in the App Connect Enterprise console for an independent integration server you should see the following messages:

   ```
   The connection agent for remote callable flows has established a connection to the Switch
    server with URL 'wss://ibm-sw-lfcxnlolwumbamsnq.eu-gb.ace.ibm.com:443/'.
   The integration server component 'agentx' has been started.
   ```

   Similarly, in the Windows Application log, you should see the following messages for an integration server (N101ISAA) under an integration node (NODEIPL01):

   ```
   ( NODEIPL01.N101ISAA ) The connection agent for remote callable flows has established
    a connection to the Switch server with URL ''wss://ibm-sw-lfcxnlolwumbamsnq.eu-
   gb.ace.ibm.com:443/''.

   A connection agent for remote callable flows has been configured on this integration server.
    The connection agent has successfully connected to the Switch server.

   No response required.
   ----------------------------------------------------------------------------------------------
   ( NODEIPL01.N101ISAA ) The integration server component ''agentx'' has been started.

   The specified component has been started and is available for use.

   No response required.
   ```

   Note: If a message flow on one on-premises integration server calls a message flow on other on-premises integration servers, and one of those integration servers also uses the callable flows technique to interact with flows in IBM App Connect on IBM Cloud, you need to copy the downloaded `agentx.json` file to all those on-premises integration servers (so that they all continue to use the same Switch server, now on cloud). For example, consider the initial on-premises scenario where a flow on server A calls a flow on server B, with both servers configured with the local `agentx.json` (to use the on-premises Switch server). Later, a flow in IBM App Connect on IBM Cloud is added to call the flow on server B, so server B needs to be configured with the `agentx.json` downloaded from the cloud (to use the on-cloud Switch server). Also, to continue to use the same Switch server as server B, server A needs to be configured with the `agentx.json` downloaded from the cloud (to use the on-cloud Switch server)

4. In IBM App Connect on IBM Cloud, you can check that the on-premises agent can connect to the Switch server in the cloud.

   You can perform the following checks:

   - Refresh the Callable flows page. This should list all the callable and calling flows on premises and on cloud; for example:
   - In the **Connect callable flows** dialog, click **Test your agent**. This should return the number of on-premises agents found.

   If the test is successful, close the dialog box.

Results

The callable flows technique can be used between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, and between integration servers in IBM App Connect Enterprise configured to use the on-cloud Switch server.

For more information about developing these message flows, see "Developing synchronously callable message flows" on page 600.

Creating destination lists

Create a list of destinations to indicate where a message is sent.

Before you begin

Read the concept topic "Message flow nodes" on page 484.

About this task

You can include a Compute node in your message flow, and configure it to create a destination list in the local environment subtree. You can then use the destination list in the following nodes:

- The MQOutput and JMSOutput nodes, to put output messages to a specified list of destinations.
- The RouteToLabel node, to pass messages to Label nodes.

For more information about accessing the LocalEnvironment subtree, destination list contents, and example procedures for setting values for each of these scenarios, see "Accessing the local environment tree" on page 1654.

For more information about how to populate destination in the LocalEnvironment subtree, and how to build JMS destination lists, see "Populating Destination in the local environment tree" on page 1656.

You might find it useful to create the contents of the destination list from an external database that is accessed by the Compute node. You can then update the destinations without needing to update and redeploy the message flow.

The use of the destination list to define which applications receive the output messages is in contrast to the publish/subscribe application model, in which the recipients of the publications are those subscribers that are currently registered with the integration node. The processing that is completed by the message flow does not have any effect on the current list of subscribers.

Validating messages

The integration node or server provides validation based on the message model for predefined messages.

Before you begin

Read the concept topics about message flows and parsers, especially "DFDL parser and domain" on page 544, "JSON parser and domain" on page 553, and "XMLNSC parser" on page 531.

About this task

Validation applies only to messages that you have modeled and deployed to the integration node or server. Specifically, the message domains that support validation are DFDL, XMLNSC, JSON, SOAP, IDOC, and MRM.

Integration nodes and servers do not provide any validation for self-defining messages. The DFDL, XMLNSC, and SOAP domains validate predefined messages directly against message model schema files. The JSON domain validates predefined messages directly against JSON schema files or OpenAPI definitions. The IDOC and MRM parsers validate predefined messages against the message dictionary generated from a message set.

Message flows are designed to transform and route messages that conform to certain rules. By default, parsers perform some validity checking on a message, but only to ensure the integrity of the parsing operation. However, you can validate a message more stringently against the message model contained in the message set by specifying validation options on certain nodes in your message flow.

You can use validation options to validate the following messages:

- Input messages that are received by an input node
- Output messages that are created, for example, by a Compute, Mapping, or JavaCompute node

These validation options can ensure the validity of data entering and leaving the message flow. The options provide you with some degree of control over the validation performed to:

- Maintain a balance between performance requirements and security requirements
- Validate at different stages of message flow completion; for example, on input of a message, before a message is propagated, or at any point in between
- Cope with messages that your message model does not fully describe

You can also specify what action to take when validation fails.

Message validation involves navigating a message tree, and checking the validity of the tree. Message validation is an extension of tree creation when the input message is parsed, and of bit stream creation when the output message is written.

Validation options are available on the following nodes:

Node type	Nodes with validation options
Input node	FileInput, FTEInput, HTTPInput, JMSInput, KafkaConsumer, MQInput, MQTTSubscribe, .NETInput, SOAPInput,TCPIPClientInput, TCPIPClientReceive, TCPIPServerInput, TCPIPServerReceiveTimeoutNotification
Output node	EmailOutput, FileOutput, FTEOutput, HTTPReply, JMSOutput, JMSReply, KafkaProducer, MQOutput, MQReply, MQTTPublish, SOAPReply, TCPIPClientOutput, TCPIPServerOutput
Other nodes	Compute, CICSRequest, DatabaseRetrieve, HTTPRequest, FileRead, JavaCompute, Mapping, MQGet, ResetContentDescriptor, SOAPRequest, SOAPAsyncResponse, Validate, XSLTransform

Validation options can also be specified on the ESQL CREATE statement and the ASBITSTREAM function. The equivalent Java API methods, **`createElementAsLastChildFromBitstream()`** and **`toBitstream()`** in MBElement, also provide validation options.

To validate input messages that are received on an input node, you can specify validation properties on the input node. The input message is then validated when the message bit stream is parsed to form the message tree.

You can also use the `Parse Timing` property of the input node to control whether the entire message is parsed and validated at this time, or whether individual fields in the message are parsed and validated only when referenced.

To validate output messages that are created by a transformation node, specify validation properties either on the node itself, or on the output node that sends the message. The validation takes place when the message bit stream is created from the message tree by the output node.

Alternatively, use a Validate node to validate a message tree at a particular place in your message flow, or use the ESQL ASBITSTREAM function in a Compute, Filter, or Database node.

A limited amount of validation occurs by default if you leave the validation settings unaltered. At this default level, an exception is thrown if one of the following statements is true:

- A data mismatch occurs; for example, the parser cannot interpret the data that is provided for the field type specified.
- The order of elements in the output message does not match the order of elements in the logical message tree (DFDL, MRM CWF, and MRM TDS fixed-length models only).

Additionally, the DFDL parser performs limited remedial action under the following circumstances:

- If mandatory content is missing, default values are supplied, if available in the schema, on output.
- If the data type of an element in the tree is CHARACTER and does not match that specified in the schema, the data type is converted on output to match the schema, if possible.

Additionally, the MRM parser performs limited remedial action under the following circumstances:

- Extraneous fields are discarded on output for fixed formats (CWF and TDS fixed-length models only).
- If mandatory content is missing, default values are supplied, if available, on output for fixed formats (CWF and TDS fixed-length models only).
- If the data type of an element in the tree does not match that specified in the dictionary, the data type is converted on output to match the dictionary definition, if possible, for all formats.

However, by using validation options you can request more thorough validation of messages. For example, you might want to validate one or more of the following conditions, and throw an exception, or log the errors:

- The whole message at the start of the message flow
- That complex elements have the correct `Composition` and `Content Validation`
- That all data fields contain the correct type of data
- That data fields conform to the value constraints in the message model
- That all mandatory fields are present in the message
- That only the expected fields are present in the message
- That message elements are in the correct order

When using validation options, it is important to understand the following behavior.

- The `Parse Timing` property, which controls whether *on-demand* parsing (sometimes called partial parsing) takes place, affects the timing of the validation of input messages, including message headers.

  For more information about the `Parse Timing` property, see Parsing on demand.
- If a message tree is passed to an output node, by default the output node inherits the validation options in force for the message tree. You can override these options by specifying a new set of validation options on the output node.
- If a message tree is passed as input to a Compute, Mapping, XSLTransform, DatabaseRetrieve, or JavaCompute node, any new output message trees that the node creates have the validation options specified by the node itself (even if the whole message is copied). You can override this behavior and specify that the messages that are created by the node inherit the validation options of the input message tree.
- (DFDL and MRM domains only) When the bit stream is written, and validation options are applied, the entire message is validated. The message tree might contain an unresolved type (for example, if a Compute node copied an unresolved type from an input message to an output message without resolving it). If such a type is encountered, a validation error occurs because it is not possible to validate

the type. To prevent this error, ensure that all unresolved types are resolved before they are copied to output messages.

- (MRM domain only) Do not select the `Truncate fixed length strings` check box because validation is done before truncation, and a fixed-length field fails validation if its length exceeds the length that is defined in the message set. For more information about the `Truncate fixed length strings` property, see Message Sets: Custom Wire Format message set properties and Message Sets: TDS Format message set properties.

For information about how you can control validation by using different properties, see Validation properties.

Changing the parser used in a message flow

If you need to change the parser that is used in a message flow, use the ResetContentDescriptor node to request that the message is reparsed by a different parser.

About this task

Use the ResetContentDescriptor node to set new message properties that are used when the message bit stream is next parsed by a subsequent node in the message flow.

For example, if the format of an incoming message is unknown when it enters a message flow, the BLOB parser is started. Later on in the message flow, you might decide that the message is predefined as a message in the MRM domain, and you can use the ResetContentDescriptor node to set the correct values to use when the message is parsed by a subsequent node in the message flow.

You can select from the following parsers:

- DFDL
- XMLNSC
- JSON
- DataObject
- XMLNS
- JMSMap
- JMSStream
- MIME
- BLOB
- XML (this domain is deprecated; use XMLNSC)
- IDOC (this domain is deprecated; use MRM)
- MRM

If you specify MRM, XMLNSC, DataObject, or IDOC as the new parser, you can also specify a different message template (message set, message type, and message format).

Whether or not the node reparses the message straight away depends on the settings of the `Parse timing` option in the node properties. `Parse timing` is, by default, set to `On Demand`, which causes parsing of the message to be delayed. For more details on controlling when the message is parsed, see Parsing on demand.

Providing user-defined properties to control behavior

User-defined properties can be set at design time, deployment time, or run time.

About this task

For example, user-defined properties can be queried, discovered, and set at run time to dynamically change the behavior of a message flow. You can use the administration REST API or IBM Integration API

to manipulate these properties, which can be used by a systems monitoring tool to perform automated actions in response to situations that it detects in the monitored systems.

Procedure

- You can use the Message Flow editor to define a user-defined property when you construct a message flow. For more information, see Message Flow editor.
- You can set user-defined properties at deployment time to configure a message flow, as described in “Configuring a message flow at deployment time with user-defined properties” on page 1749.
- You can use the administration REST API to modify user-defined properties on a message flow dynamically at run time, as described in “Setting message flow user-defined properties at run time by using the administration REST API” on page 436.
-

Defining message flow content

You can define message flow content by adding and configuring message flow nodes and connecting them to form flows.

About this task

When you create a message flow, the editor view is initially empty. You must create the contents of the message flow by using a combination of the following tasks:

- “Adding a message flow node” on page 618
- “Adding a subflow” on page 623
- “Renaming a message flow node” on page 624
- “Configuring a message flow node” on page 624
- “Connecting message flow nodes” on page 636
- “Inserting nodes into existing message flows” on page 639
- “Adding a bend point” on page 640
- “Aligning and arranging nodes” on page 642

While you are developing message flows, you might want to record notes about flow development or particular nodes in the flow:

- “Adding annotations to a message flow or node” on page 643
- “Editing annotations on a message flow or node” on page 644
- “Copying annotations on a message flow or node” on page 645
- “Showing and hiding annotations on a message flow or node” on page 646
- “Deleting annotations from a message flow or node” on page 646

When you finalize the contents of the message flow, you might also need to complete the following tasks:

- “Removing a message flow node” on page 636
- “Removing a node connection” on page 639
- “Removing a bend point” on page 641

Using the node palette

The node palette contains all the built-in nodes, which are organized into categories.

Before you begin

Read the following concept topic “Message flow node palette” on page 486.

About this task

You can add the nodes that you use most often to the Favorites category by following the instructions in “Adding nodes to the Favorites category on the palette” on page 617.

You can change the palette preferences in the IBM App Connect Enterprise Toolkit. The changes that you can make are described in the following topics.

Procedure

Changing the palette layout

You can change the layout of the palette in the Message Flow editor.

Procedure

1. Switch to the Integration Development perspective
2. Right-click the palette to display the pop-up menu.
3. Click **Layout**.
4. Click one of the available views:

   Columns
   Displays named icons in one or more columns. Change the number of columns by clicking on the right edge of the palette and dragging.

   List
   Displays named icons in a single-column list. The list view is the default layout.

   Icons Only
   Displays a list of icons only.

   Details
   Displays descriptions of the icons.

Changing the palette settings

How to use the **Palette Settings** dialog box.

About this task

Change the palette settings in the Message Flow editor using the **Palette Settings** dialog box.

Procedure

1. Switch to the Integration Development perspective.
2. Right-click the palette to display the pop-up menu.
3. Click **Settings**.
   The **Palette Settings** dialog box opens.

4. Use the dialog to change the appropriate setting:
   - Click **Change** to change the font on the palette.
   - Click **Restore Default** to restore the default palette settings.
   - In the **Layout** list, click the appropriate radio button to change the palette layout. (See "Changing the palette layout" on page 615 for more information.)
   - Select **User large icons** to toggle between large and small icons in the palette.
   - In the **Drawer options** list, click the appropriate radio button to change the way that drawers are handled in the palette. A drawer is a container for a list of icons, such as the **Favorites** drawer on the Message Flow editor's palette, or the **Entity** drawer on the Integration node Topology editor's palette.

Customizing the palette

If you customize the message flow node palette, you can make it easier to find the nodes that you use most often.

About this task

This saves time and on-screen space. For example:

- Change the order of the drawers in the palette so that the ones that you use most often are at the top.
- Hide any drawers that you do not use, to save on-screen space.
- Pin open the drawers that contain the nodes that you use most often.
- Create your own drawers to hold user-defined nodes that you create.

Customize the palette for the Message Flow editor using the Customize Palette dialog box:

Procedure

1. Switch to the Integration Development perspective.
2. Right-click the palette, then click **Customize**. The Customize Palette dialog box opens.
   - To change the order of entries and drawers in the palette, click the appropriate item in the list to highlight it, then click **Move Down** or **Move Up**. You cannot move any category above the Favorites category.
   - To hide an entry or drawer, click the appropriate item in the list to highlight it, then select the **Hide** check box.
   - To create a new separator, click **New** > **Separator**.
   - To create a new drawer:
     a. Click **New** > **Drawer**.
     b. Type a name and description for the drawer.
     c. If required, select the **Open drawer at start-up** check box.
     d. If required, select the **Pin drawer open at start-up** check box.
3. Click **OK** or **Apply** to save your changes.

Results

You have customized the message flow node palette.

Adding nodes to the Favorites category on the palette

The nodes on the palette are organized in categories. The first category is Favorites, which is usually empty. You can drag the nodes that you use most often to the Favorites category.

Before you begin

Read the concept topic about the message flow node palette.

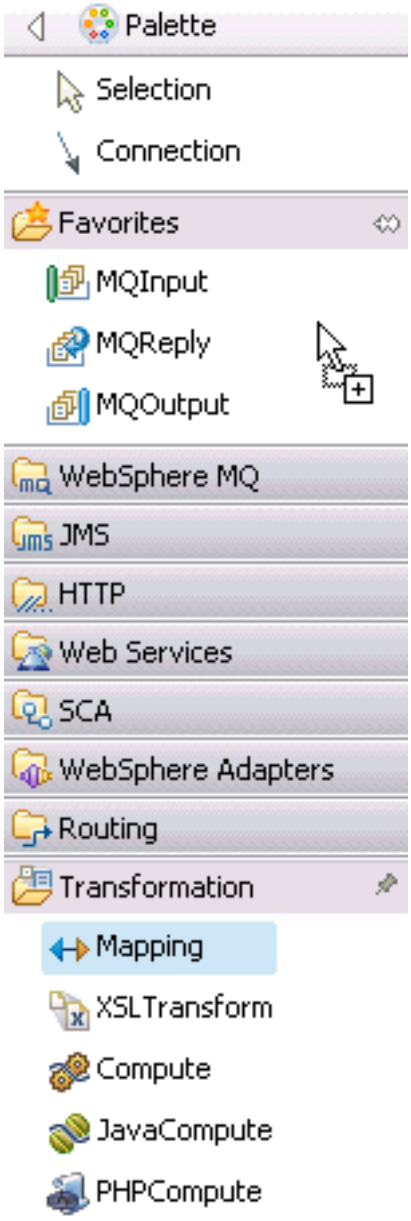
About this task

The following steps describe how to drag nodes into the Favorites category.

Procedure

1. Switch to the Integration Development perspective.
2. On the palette, open the Favorites category.
3. On the palette, open the category that contains the node that you want to add to the Favorites category.

4. Use the mouse to drag the node into the Favorites category, as shown in the following example:



Results

Alternatively, right-click the palette and choose the appropriate option to add or remove nodes from the Favorites category.

Adding a message flow node

When you have created a message flow, add nodes to define its function.

Before you begin

- Read the concept topic about message flow nodes, see "Message flow nodes" on page 484.
- Create a message flow or open an existing message flow; see "Creating a message flow" on page 574 and "Opening an existing message flow or subflow" on page 577.
- If you want to use a user-defined node or connector in your message flow, you need to install it in the IBM App Connect Enterprise Toolkit. For more information, see "Installing a user-defined node" on page 620 or "Using a connector in a message flow" on page 620.

About this task

To add a node to a message flow:

Procedure

1. Open the message flow with which you want to work.
2. Open the Palette.
   - Hold the mouse pointer over the palette bar while it is in collapsed mode. The palette bar expands. When you move the mouse pointer away from the palette bar, it collapses again.
   - Click the **Show Palette** icon at the top of the palette bar. The palette bar expands and it remains expanded when the mouse is moved away from the palette bar. To collapse the palette bar again, click the **Hide Palette** icon at the top of the palette bar while it is in expanded mode.
3. Click **Selection** above the palette of nodes.
4. Decide which node you want to add: a built-in node or a user-defined node.

   You can select any of the nodes that are displayed in the node palette, but you can add only one node at a time.

   Nodes are grouped in categories according to the function that they provide. To see descriptions of the nodes in the palette, either hold the mouse pointer over a node in the palette, or switch to the Details view by following the instructions in "Changing the palette layout" on page 615.
5. Drag the node from the node palette onto the canvas.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you rename the node, the name that you choose must be unique in the message flow. If you do not change the default name at this time, you can change it later, see "Renaming a message flow node" on page 624. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.
6. Repeat steps "4" on page 619 and "5" on page 619 to add further nodes.
7. You can also add nodes from other flows into this flow:
   a) Open the other message flow.
   b) Select the node or nodes that you want to copy from the editor or outline views, and press Ctrl+C or click **Edit** > **Copy**.
   c) Return to the flow with which you are currently working.
   d) Press Ctrl+V or click **Edit** > **Paste**.

      This action copies the node or nodes into your current flow. The node names and properties are preserved in the new copy.

Results

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow, and you can configure their properties.

What to do next

Now you can configure the nodes, see "Configuring a message flow node" on page 624.

Installing a user-defined node

Install a user-defined node for use to develop message flows.

About this task

You can develop message flows by using the user-defined node in the same way as the built-in nodes. Before you can use a user-defined node, you must install it into your IBM App Connect Enterprise Toolkit. A user-defined node is installed differently depending on how it was packaged.

Procedure

1. Install the user-defined node project:
   - If the user-defined node was packaged as plug-in JAR files, copy your JAR files into the dropins folder in your IBM App Connect Enterprise Toolkit installation location, and run the **`ace toolkit -clean -initialize`** command to pick up the new files.
   - If the user-defined node was packaged as an update site, install the plug-ins from the update site.
2. Restart your IBM App Connect Enterprise Toolkit. The user-defined nodes are displayed in the palette in the Message Flow editor under the category that you specified for the user-defined node project.

What to do next

If you have created a user-defined node in Java or C that uses a custom property compiler, you must also install the user-defined node plug-in on the integration nodes or independent integration servers onto which you want to deploy the user-defined node; see the following topics:

- "Installing user-defined extension runtime files on an integration server" on page 2456
- "Installing user-defined extension runtime files on an integration node" on page 2455

Message sets packaged with a user-defined node

Adding message sets to an update site.

About this task

Message sets are automatically packaged into user-defined node JAR files and update sites. If you drag a user-defined node from the palette onto the canvas, the IBM App Connect Enterprise Toolkit detects whether the required message sets are available in the current workspace.

Procedure

1. If the required message sets are unavailable in the current workspace, a warning is displayed in the **Problems** tab, and the **Import Required Message Sets** window opens.
2. To import the message set from the user-defined node plug-in, click **Import selected message sets**, or you can import the message sets from another source.

   If the message set is available in the same integration server at execution time, you do not have to import the message sets.

Using a connector in a message flow

The capabilities of a connector are exposed to a message flow in a user-defined node. Before you can use that user-defined node, you must install it into your IBM App Connect Enterprise Toolkit.

About this task

Connectors can be used in IBM App Connect Enterprise to interact with applications or data sources. A connector developer creates a connector for the runtime environment and a user-defined node for the IBM App Connect Enterprise Toolkit. The connector developer provides this user-defined node in a plug-in JAR file. You can develop message flows by using the user-defined node in the same way as the built-in nodes.

Procedure

To install the user-defined node into your IBM App Connect Enterprise Toolkit, complete the following steps:

1. Copy the user-defined node plug-in JAR file into the `dropins` folder in your IBM App Connect Enterprise Toolkit installation location: *`install_dir`*`/tools/dropins`.

   For example, the default installation directory on Windows is `C:\Program Files\IBM\IIB\`*`version`*`\`.
2. To ensure that the new files are available to the IBM App Connect Enterprise Toolkit, restart the Toolkit.

Results

The user-defined node for your connector is displayed in the palette in the Message Flow editor. If the connector developer specified a category in the user-defined node project, the user-defined node is displayed beneath that category.

What to do next

Add the user-defined node to a message flow by following the instructions in "Adding a message flow node" on page 618.

Adding a message flow node by using the keyboard

You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

Before you begin

- Ensure that you have created or opened a message flow. For more information, see "Creating a message flow" on page 574 or "Opening an existing message flow or subflow" on page 577.
- Read the concept topic, "Message flow nodes" on page 484.

Procedure

1. Open the message flow to which you want to add a node.
2. Open the Palette view or the Palette bar.
3. Select a node in the Palette view or Palette bar by using the up and down arrows to highlight the node that you want to add to the canvas.
4. Add the node to the canvas by using one of the following methods:
   - Press **Alt + L**, then press **N**.
   - Press **Shift + F10** to open the pop-up menu for the Palette, and press **N**.

   The node that you selected in the Palette bar or Palette view is placed on the canvas in the Editor view.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later by following the instructions in "Renaming a message flow node" on page 624. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput. If you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

Dragging a resource from the Application Development view

Drag a node or a related resource into the Message Flow editor.

Before you begin

Complete the following tasks:

- Create or open a message flow, as described in “Creating a message flow” on page 574“Creating a message flow” on page 574 and “Opening an existing message flow or subflow” on page 577.
- Read concept information in “Message flow nodes” on page 484.

About this task

To create a node, drag a resource from the Application Development view to an empty canvas. To modify an existing node, drag a resource onto that node. The following resources are supported:

- An Adapter file
- An IBM Integration Bus SCA definition file
- A database service
- An ESQL file
- A Java file
- An MQ service
- A subflow
- A WSDL file
- An XSL file

Procedure

1. Open the message flow with which you want to work.
2. Drag one of the supported resources from the Application Development view onto the canvas.
   - If you drop the resource on an empty canvas, a node is created and configured automatically.

     The following table shows the results when you drag a resource from the Application Development view onto an empty canvas:

Resource	Node created	Property set
Adapter file	A PeopleSoftInput node, SAPInput node, or SiebelInput node is created.	`Adapter component`
CORBA IDL file	A CORBARequest node is created.	`IDL file`
ESQL file	A Compute node is created.	`ESQL Module`
Java file	A JavaCompute node is created.	`Java Class`
WSDL file	A SOAPInput node, SOAPRequest node, or SOAPAsyncRequest node is created.	`WSDL file name`
XSL file	An XSLTransform node is created.	`Stylesheet`

   - If you drop the resource onto an existing node, the relevant node property is updated with the name of the resource file. For example, if you drop a Java file onto a JavaCompute node, the `Java Class` property is set to the class name of the Java file that you are dropping. If you drop an ESQL file over any node that uses ESQL, such as a Database node, the `ESQL Module` property is set. Further examples include dropping an MQ Service onto an MQ node to configure it, or dropping a database service onto a Compute node to configure it.

Adding a subflow

In a message flow, you can include an embedded message flow, also known as a *subflow*. For example, you might define a subflow that provides error handling, and include it in a message flow connected to a failure terminal on a node that can generate an error in some situations.

Before you begin

To complete this task, you must have created your subflow in either a `.subflow` file, or in a `.msgflow` file. You must also have created a message flow in which to insert the subflow.

- For more information about creating a `.subflow` file, see "Creating a subflow" on page 575.
- For more information about creating a `.msgflow` file, see "Creating a message flow" on page 574.
- For more information about the differences between subflows defined in `.subflow` files and subflows defined in `.msgflow` files, see "Subflows" on page 488.

About this task

You can embed subflows into your message flow if either of the following statements is true:

- The subflow that you want to embed is defined in the same application, library, or integration project as the message flow.
- The subflow is defined in a different library, and you have specified the dependency of the current application, library, or integration project on that other library.

If you want to embed subflows into other subflows, be aware of the following information:

- You can embed subflows that are defined in `.subflow` files into subflows that are defined in `.subflow` files and `.msgflow` files.
- You can embed subflows that are defined in `.msgflow` files into subflows that are defined only in `.msgflow` files.
- A subflow in a shared library can contain another subflow in the same library or in another shared library.

Subflows in shared libraries cannot be in the default broker schema.

When an application or shared library references other shared libraries, all the subflows for a broker schema must be in a single container. Subflows for a broker schema must not be in both an application (or shared library) and a shared library that is referenced by that application (or shared library). Subflows for a broker schema must not be in two or more shared libraries that are referenced by a single application or shared library. All the subflows in a broker schema must be either in the main application or shared library, or in a single referenced shared library.

Procedure

To add a subflow to a message flow or subflow, complete the following steps:

1. Open the message flow or subflow in which you want to embed the subflow.
2. To add a subflow, drag the message flow or subflow that you want to add from the Application Development view to the editor.

   Alternatively, click **Flow** > **Add subflow**, then select the flow that you want to add from the list.

   The embedded subflow is shown in the Message Flow editor as a single node with the terminals that represent the Input and Output nodes that you have included in the subflow.
3. Connect the subflow node to one or more of the nodes in the main message flow or subflow.

   For more details, see "Connecting message flow nodes" on page 636.
4. To add and connect further subflow nodes, repeat steps "2" on page 623 and "3" on page 623.
5. To work with the contents of the subflow, double-click the subflow icon.

   The subflow opens in the Message Flow editor.

What to do next

Renaming a message flow node

You can change the name of any type of message flow node (a built-in node, user-defined node, or subflow node) to reflect its purpose.

Before you begin

Complete the following tasks:

- Create a message flow, as described in "Creating a message flow" on page 574.
- Read the concept topic, "Message flow nodes" on page 484.

About this task

When you first add a message flow node to the canvas, the editor automatically assigns a name to the node; the name is highlighted, and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later, as described in this topic. For example, you might include a Compute node to calculate the price of a specific part in an order; you could change the name of the node to Calculate_Price.

When you rename a node, use only the supported characters for this entity. The editor prevents you from entering unsupported characters.

To rename a node:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow with which you want to work.
3. You can rename a message flow node in three ways:
   - Right-click the node and click **Rename**. The name is highlighted; enter a name of your choice and press **Enter**.
   - Click the node to select it, then click the name of the node so that it is highlighted; enter a name of your choice and press **Enter**.
   - Click the node to select it, then on the Description tab of the Properties view, enter a name of your choice in the Node name field.

   The name that you enter must be unique in the message flow.

What to do next

If you generate ESQL code for a Compute, Database, or Filter node, the code is contained in a module that is associated with the node. The name of the module in the ESQL file must match the name specified for the module in the *ESQL Module* property of the corresponding node. You can modify the module name, and change it from its default value (which is the name of the message flow, concatenated with the name of the node with which the module is associated). However, ensure that the module in the ESQL file matches the node property.

Configuring a message flow node

When you have included an instance of a node in your message flow, you can configure its properties to customize how it works.

Before you begin

- Read the concept topic about message flow nodes

- Add a node

Viewing the properties of a node

About this task

To view a node's properties:

Procedure

1. In the Message Flow editor, open the message flow with which you want to work.
2. To open the Properties view, right-click a node and click **Properties**.

   The selected node's properties are displayed. You can edit the properties.

Editing the properties of a node

About this task

Properties are organized into related groups and displayed on tabs. Each tab is listed on the left of the Properties view. Click each tab to view the properties that you can edit.

Procedure

- Every node has at least one tab, **Description**, where you can change the name of the node and enter short and long descriptions. The description fields are optional because they are used only for documentation purposes.
- If a property is mandatory (that is, one for which you must enter a value), the property name is marked with an asterisk, as shown in the following example:

  ```
  Queue Name* _______________________________
  ```

- Many nodes have properties that require XPath expressions, typically to identify the location of a particular resource.

  For example, for the MQInput node, you can specify the location of the security token and password by using either ESQL or XPath expressions. For help in constructing an XPath expression, you can open the XPath Expression Builder by clicking **Edit** next to each XPath property. For more information about using XPath expressions, see “Using XPath” on page 626.

What to do next

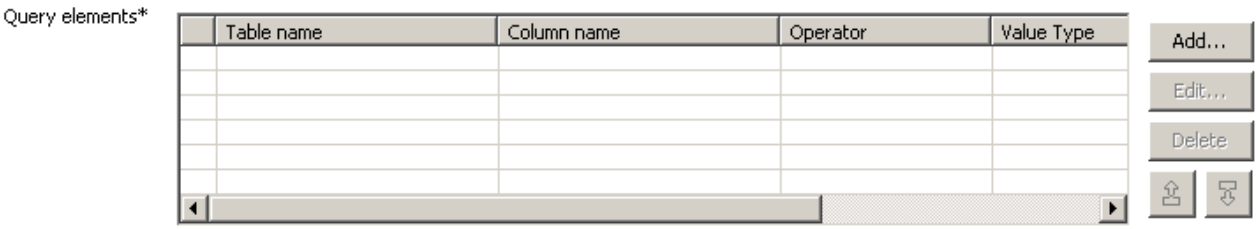
For detailed information about how to configure individual built-in nodes, see the reference topic for the relevant node (see Built-in nodes).

If you have included a user-defined node, refer to the documentation that came with the node to understand if, and how, you can configure its properties.

Editing complex properties

About this task

A complex property is a property to which you can assign multiple values. Complex properties are displayed in a table in the Properties view, where you can add, edit, and delete values, and change the order of the values in the table. This example shows the `Query elements` complex property of the DatabaseRoute node.



Procedure

- To add a value to a complex property, click **Add**, enter the required fields in the dialog box that opens, then click **OK**.

  The values appear in the table. Repeat this step to enter as many values as are required.
- To edit a value, click any element in a row, click **Edit**, edit any of the values in the dialog box, then click **OK**.
- To delete a value, click any element in a row and click **Delete**.

  The entire row is deleted.
- To change the order of values in the table, click any element in a row and click the up icon  or down icon  to move the row.

Overriding properties at deployment time

About this task

You can override some node property values when you deploy a message flow. These property values are known as configurable properties, and you can use them to modify some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager and data source information.

Even though you can set values for configurable properties at deployment time, you must set values for these properties within the message flow if they are mandatory. Each built-in node reference topic contains a table of properties, which identifies the configurable and mandatory properties.

You can override message flow node properties at run time by deploying policies, either directly to an integration server or in a BAR file with your message flows. Alternatively, you can put policies in the `overrides` sub-directory of the integration server work directory. For more information, see "Overriding properties at run time with policies" on page 324.

What to do next

Next: connect the nodes.

Using XPath

XPath provides an alternative method to ESQL for entering expressions in the property fields of specific built-in nodes.

About this task

In addition to ESQL as a message transformation language, IBM App Connect Enterprise supports an alternative expression grammar in property fields. For more information, see ESQL-to-XPath mapping table.

You can use ESQL or XPath expressions in built-in nodes, within your message flows, to query or update message trees that are specified as accessible, and that you expect to be processed by a given node.

This section provides information on:

XPath overview

The XML Path Language (XPath) is used to uniquely identify or address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from any part of the document, such as an element or attribute (referred to as a *node* in XML) in it. XPath can be used alone or in conjunction with XSLT.

Some of the built-in nodes provided in the IBM App Connect Enterprise Toolkit can use XPath expressions to specify the part of a message that is processed by the node. For example, you can use an XPath expression to identify fields in a message and determine if they match a specified value, or to set the field value by updating it with the results of a database query.

You can use XPath 1.0 path expressions in your flow to access specific parts of an incoming message, create or locate parts of an outgoing message, and perform complex message processing that might involve values present in message trees accessible by a node so that you can transform, filter, or retrieve values from a message.

For example, the Route node applies XPath 1.0 general expressions to the content of message trees associated with the incoming message assembly for this node. Following evaluation of an expression the result is cast as a Boolean (true or false) result, and this in turn is used to determine if a copy of the incoming message is routed down an output terminal associated with the processed expression.

If you have XML schema or DFDL message definition files (.xsd), or Message Sets (.mxsd) present in your workspace, any elements, attributes or data types defined in such definitions can be loaded into the Data types viewer and selected to automatically generate a path expressions mapping to the definition concerned.

Equally, depending on the XPath expressions supported by the property concerned, you can select XPath functions and operators to include in an expression, or you can build your own expressions manually.

The **Data types viewer** contains a list of variables relating to trees that can be accessed by expressions for the node property concerned.

For example, `$InputRoot` gives access to the incoming message tree. The fixed format of standard folders you can expect to exist under this tree, for example, `Properties` and `MQMD` are described without the need to import an `.mxsd` definition for them. These structures can be navigated in the viewer and, on selection of an element within them, a path expression that maps to the element in question is built automatically through the XPath 1.0 language.

For further information on XPath 1.0 see W3C XPath 1.0 Specification.

You can use the XPath Expression Builder to visually build XPath expressions to set the relevant properties in your nodes. You launch the XPath Expression Builder from buttons located along side property fields present in the Properties viewer, for those nodes that support the use of XPath expressions as property values.

The XPath files in IBM App Connect Enterprise are supplied in three property editors; see XPath property editors for more details.

The XPath editor supports both content-assist directly on the text field and also an **Edit...** button that launches the XPath builder dialog. The dialog gives you a larger area in which to build your XPath expressions.

The content assist based control contains two different proposal lists in the following order:

1. Nodes and Variables

2. Functions and Operators

The node and variable proposals are displayed the first time that you use the XPath editor. In this view, the status bar reads `Press` **`Ctrl+Space`** `to show Function and Operation Proposals`.

Pressing **`Ctrl+Space`** when you are in the function and operator level proposals selects the Node and Variable proposals.

Note: In the Graphical Data Mapping editor, you can use XPath 2.0 functions and expressions. You can use XPath functions to define graphical transformations in your message. You can use XPath expressions to define conditions for your transformations.

Namespace support
The XPath Expression builder provides qualified support for namespaces.

The XPath Expression builder dialog contains a namespace settings table that:

- Supports simplified expressions that enable qualified namespace matching at run time
- Can be auto-generated based on imported schema definitions and generated expressions (based on selections made in the dialog when you build an expression)
- Allows you to add your own entries to the table

The table encapsulates deployable data passed to the runtime environment, as part of the nodes attribute data, and is used by the node to modify expressions through prefix-to-URI substitution. The final expressions support namespace matching, because they are processed against a target tree when employed by their associated message processing engine, that is, the XPath 1.0 runtime engine or ESQL runtime engine.

When you enter an ESQL field reference expression in a read-only or read-write path field, or an XPath 1.0 path expression in a read-only or read-write path field, or a general expression field (general expressions can contain zero or more path expressions), IBM App Connect Enterprise understands the language from the syntax you use.

XPath is the default for general expression fields that are validated by ensuring they conform to the XPath 1.0 grammar. For path expression fields XPath is assumed if the expression is valid and begins with a $ sign.

The language you can use is dictated by the property editor currently in use for a node's property field.

Namespace prefixes are used in an XPath or ESQL expression to make the statements shorter and easier to understand, while still supporting the ability to qualify an element name match by also matching on its associated namespace URI.

For example, consider the following message, where namespace prefix b is overridden through an inner declaration:

```
<b:a xmlns:b='xyz'>
 <!-- the namespace of elements 'a' and 'c' using prefix 'b'  is xyz -->
 <b:c>
  <b:d  xmlns:b='qrs'>
   <!-- the namespace of elements 'd' and 'e' using prefix 'b' is now qrs -->
   <b:e>100</b:e>
  </b:d>
 </b:c>
</b:a>
```

Note that the scope of a namespace declaration declaring a prefix extends from the beginning of the start tag in which it appears to the end of the corresponding end tag, excluding the scope of any inner declarations with the same namespace prefix. In the case of an empty tag, the scope is the tag itself: >.

To navigate to element e in the above message use the following human-readable XPath expression:

```
/b:a/b:c/b2:d/b2:e
```

Note, that to prevent the auto-generated prefix to the URI map produced in the expression dialog overloading the same prefix (in this case b), the inner b prefix is appended with a numeric value to distinguish it from the outer b prefix. This strategy is repeated for each prefix name clash.

This notation is similar to the equivalent human-readable ESQL expression:

```
Root.b:a.b:c.b2:d.b2:e
```

To support namespace prefixes within expressions, the XPath Expression Builder Dialog automatically generates a prefix to a URI namespace settings table (based on the content of imported schema definitions, through which expressions are generated) .

Without the use of namespace prefixes to URI mapping data in this table, the runtime environment would be forced to take a less efficient approach, where portable but verbose XPath expressions would be required by it to provide namespace matching support.

The previous expression:

```
/b:a/b:c/b2:d/b2:e
```

would take the form:

```
/*[namespace-uri()='xyz' and local-name()='a']/*[namespace-uri()='xyz'
   and local-name()='c']/*[namespace-uri()='qrs' and
   local-name()='d']/*[namespace-uri()='qrs' and local-name()='e']
```

XPath Expression Builder
You can launch the XPath Expression Builder from most property fields that support, or expect, XPath expressions as a value that can be entered into the field.

The use of the XPath Expression Builder is optional, in that it is an aid to you in developing message flow applications. The XPath Expression Builder helps you to construct message processing expressions in either XPath or ESQL. You are free to enter expressions by hand, or use the XPath Expression Builder to help construct such expressions.

The XPath Expression Builder does not support use of the $Body variable. You can use the $Body variable when you enter an expression by hand, but the XPath Expression Builder and associated validation in the IBM App Connect Enterprise Toolkit do not support it. Use the $Root variable instead.

You can populate the fields, regardless of the state of the node; that is, whether the node is detached or connected, or fully, partially, or completely unconfigured.

You launch the XPath Expression Builder from a button in the following locations:

- Table cells, located to the right of the text entry field within the cell.
- **Add** or **Edit** dialog boxes used to construct rows in tables, located to right of the property field concerned.
- Tabs in the property viewer for a node, to the right of a property field.

Variables (or in ESQL terminology, correlation names) provide a list of all message tree start points that are applicable to the property field from which the dialog was launched.

If a field is a read-only or a read-write path field, expressions must start with such a variable to indicate which tree in which message assembly the path expression is mapping to.

XPath variable names map to existing correlation names found in ESQL field reference expressions, but to conform to the ESQL grammar they are designated as variable references by prefixing them with the dollar ($) character.

For example:

ESQL
```
Root.XMLNSC.CUST_DETAILS.NAME
```

XPATH
```
$Root/XMLNSC/CUST_DETAILS/NAME
```

The variable indicates to which tree and where in that tree the expression is anchored.

The XPath Expression Builder dialog box supports validation, which you can turn off on the XPath preferences page by clearing the **Validate when creating XPath expressions** check box.

If you select the `$Root` or `$Body` variables and create an expression that refers to the body of the message, the XPath expression contains the message element. This expression is correct for message bodies owned by the XMLNSC, XMLNS, XML, and DataObject domains.

For message bodies that are owned by the SOAP, MIME, MRM, and IDOC domains, you must remove the message element from the expression.

For example, the XPath expression `$Body/my_message/my_field` is correct for XMLNSC, but must be changed to `$Body/my_field` to be correct for MRM.

Views

There are three main views when functions are supported.

Whether a view is displayed, and what is displayed in it, depends on what type of property editor you have used to launch the dialog, and its tailored settings; for example, for path type fields you do not see a functions pane. The operators that are supported can change as can the list of applicable variables.

Data Types Viewer
This view shows the different schema types, elements, and attributes that you can use within the XPath expression that you are creating, as well as the allowable variable references.

XPath Function
This view shows four main top level categories, which are:

String
This category corresponds to the description in the XPath 1.0 specification of `section-String-Functions`.

Boolean
This category corresponds to the description in the XPath 1.0 specification of `section-Boolean-Functions`.

Numeric
This category corresponds to the description in the XPath 1.0 specification of `section-Number-Functions`.

Nodeset
This category corresponds to the description in the XPath 1.0 specification of `section-Node-Set-Functions`.

For information on the format of XPath 1.0 expressions see the W3C XPath 1.0 Specification.

Operators
This view shows a list of all of the available operators that you can use within the given XPath expression.

Namespace settings

If you expand **Namespace settings** in the XPath Expression Builder dialog you see a table of Prefix and Namespace pair strings. This table is automatically updated when XPath expressions are created. If the default prefix generated is not what you want, you can change it by clicking **Change Prefix**.

To add a prefix and namespace map entry click **Add** and complete the fields in the dialog.

To edit or delete an entry in the table, select the item and click **Edit** or **Delete** respectively.

Edit opens another field dialog allowing you to change the prefix and namespace.

For information about the preferences supplied with the XPath editor, see “XPath editor preferences” on page 631.

XPath editor preferences
This topic describes the possible options available to you when you use the XPath editor.

The following lists describe the custom preferences used in the XPath editor.

- Validation option:

  Validation when building XPath expressions
  Default: checked.

  This option is used to perform validation each time you update the XPath expression. As validation requires re-parsing the expression against the XML Schema document you can turn this option off, for example, if you are dealing with very large complex XPath expressions.

- Content Assist display options:

  Show XML Schema model groups
  Default: not checked.

  This option allows you to view XML schema model groups, or not.

  Show type in XML Schema tree
  Default: checked.

  This option allows you to view the `<type name>` in both the Content Assist view and the XPath expression builder, or not.

  Show function parameters
  Default: checked.

  This option allows you to have function parameters shown, or not.

  Show function return type
  Default: checked.

  This option allows you to have function return types shown, or not.

  Show content assist description
  Default: checked.

  This option allows you to view the description of a given selected entry in the Content Assist view, or not..

- Auto-Activation option:

  Enable auto activation
  Default: checked.

  When this option is active, the Content Assist field appears whenever the cursor is after one of the following characters:

  – / - Forward slash character
  – [- left bracket character
  – (- Left parentheses character
  – , - Comma character

  You set the delay time, before the Content Assist field appears, in the **Auto activation delay** field. The time is in milliseconds and the range is a positive number between zero and 9999.

- Content assist color options:

  This preference allows you to customize the background and foreground colors for the Content Assist fields. The default background color is (red, green, blue - 254, 241, 233) and the default foreground color is (red, green, blue - 0, 0, 0) - black.

XPath expressions supported by default
XPath supports a number of expressions by default.

The following expressions are supported:

Read-only fields

$Root, $Body, $Properties, $LocalEnvironment, $DestinationList, $ExceptionList, $InputRoot , $InputBody, $InputProperties, $InputLocalEnvironment, $InputDestinationList, $InputExceptionList, $Environment.

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadOnlyPropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList'
$Environment
```

Read-write fields

$InputRoot , $InputBody, $InputProperties, $InputLocalEnvironment, $InputDestinationList, $InputExceptionList, $OutputRoot , $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList, $Environment.

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathReadWritePropertyEditor", "InputRoot ,
 InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
 InputExceptionList"
```

restricts the XPath field to support only:

```
$OutputRoot, $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList,
 $Environment
```

Expression fields

$Root, $Body, $Properties, $LocalEnvironment, $DestinationList, $ExceptionList, $InputRoot , $InputBody, $InputProperties, $InputLocalEnvironment, $InputDestinationList, $InputExceptionList, $OutputRoot , $OutputLocalEnvironment, $OutputDestinationList, $OutputExceptionList, $Environment.

To exclude variables for a node property from the default list, specify a comma separated string of variables. For example, specifying:

```
"com.ibm.etools.mft.ibmnodes.editors.xpath.XPathPropertyEditor", "InputRoot ,
InputBody, InputProperties, InputLocalEnvironment, InputDestinationList,
InputExceptionList, OutputRoot , OutputLocalEnvironment, OutputDestinationList,
OutputExceptionList"
```

restricts the XPath field to support only:

```
$Root, $Body, $Properties, $LocalEnvironment,  $DestinationList, $ExceptionList,
 $Environment.
```

Creating XPath expressions

Many message flow nodes provide properties that can be specified using an XPath 1.0 expression, for example properties to set data locations for Monitoring or Security where this language is used to form

a path expression to locate incoming message body elements. The Mapping node supports XPath 2.0 functions and expressions.

About this task

Other less common node property fields support the entry of general XPath 1.0 expressions that support a wider range of the language to perform more complex evaluations in the integration node's XPath 1.0 runtime engine.

Note: In the Graphical Data Mapping editor, you can use XPath 2.0 functions and expressions. You can use XPath functions to define graphical transformations in your message. You can use XPath expressions to define conditions for your transformations.

The XPath Expression builder provides a tree view of a message, and supports the automatic generation of an XPath 1.0 path expression, through the selection of an element within the tree.

The **Schema Viewer** section provides a tree view of the input message. To visually build your XPath expression follow these steps:

Procedure

1. Add the relevant node to your message flow
2. In the Properties viewer, enter the correlation name, or press Ctrl + Space to use content assist, or press Edit to use the Expression editor.

   Content assist is also invoked by simply typing $ in cell-based property fields. See "Correlation names" on page 519 for further information on correlation names.
3. Expand the tree, navigate to the field for which you want to build an expression, and click to select it.

   A field is either an element, or an attribute.

   Double-click the field to add it to the XPath expression. You can also drag fields, functions, and operators to the desired location in the XPath expression when using the XPath Expression builder.
4. To set conditions, enter them as you would a normal XPath Expression.

Results

The complete XPath expression is shown either:

- In the XPath Expression pane if you are using the XPath Expression builder.

  The Expression builder dialog is an optional aid for generating expressions that, when complete, form the value in a node's property field.

  If you do not use the Expression builder dialog, the expressions entered manually are validated using the property editor.
- In the Property field if it is in the node itself.

Messages are displayed at the top of the XPath Editor window to alert you to the fact that a path or expression you have entered is not valid.

Note: The editor does not prevent you from entering, and saving, an expression that is not valid.

Example

Here is the XPath expression built in the XPath Expression Builder to filter the Employee business objects for all employees who are managers: `$Root/XMLNSC/getEmployeeInfo/Emp[isManager=true()]`.

- `$Root/XMLNSC/`: The body section of the message; that is, the last child of root. This example assumes that the XMLNSC domain is being used.
- `/getEmployeeInfo`: The name of the operation in the interface.
- `/Emp`: The name of the input message type.
- `[isManager=true()]`: Checks whether the `isManager` field is set to true.

In this case the same expression works for both request and response flows, because the input and output messages for the operation are identical.

For more information on XPath 1.0, see W3C XPath 1.0 Specification.

Selecting the grammar mode
The grammar mode allows you to use only a restricted set of expressions, in either XPath or ESQL, and checks whether the syntax you have entered is valid.

About this task

IBM App Connect Enterprise supports the following field categories:

- Read-only path field
- Read-write path field
- Expression field

Each of these field types can be either fixed or mixed language, that is, ESQL, XPath, or either.

If you use XPath syntax, and the expressions are not supported for the property you are using, the syntax is rejected during the validation process.

ESQL and XPath have similar restrictions on the syntax that is permitted for the first two of these field types. There are restrictions to the expression fields as well, but as this type of field supports general expressions that can be used in either language, the range of syntax available is greater than in the first two.

IBM App Connect Enterprise uses code assistance in the grammar management of XPath 1.0 to validate the syntax of expressions you enter. This assistance is always available, regardless of the grammar mode you are using.

By default, you are operating in the restricted grammar mode.

Code assistance enables you to construct syntactically correct expressions but it does not validate those expressions. Validation is performed by property editors in which such expressions are entered.

If you attempt to use an expression that is not valid, the property editor marks it as such, either from a syntax or schema validation perspective.

You receive error or warning messages depending on the preference choices you set in **Windows>Preferences>Broker Development>XPath>Validation** .

If, under the above validation settings, particular checks are to be marked as errors, error markers are shown in the problems viewer. This behavior results in a message flow being marked as broken, and it cannot then be imported into, or compiled in, a deployable BAR file using the BAR editor.

If you want to use the appropriate unrestricted grammar to input a specific field type, property editors do not force restricted forms of ESQL or XPath 1.0 expressions for such fields that expect them. Instead, you can enter the full range of syntax in the context of the field category concerned, namely, path or general expression, without the validation checks being applied. This means that if you need to, you can deploy the full range of syntax supported by the ESQL or XPath 1.0 runtime environment. Note, however, that such expressions might not be in a form that can be converted to the other language.

To use unrestricted grammar, carry out the following procedure:

Procedure

1. Click **Window -> Preferences** and expand Integration Development.
2. Expand **XPath** and click **Grammar**.
3. Clear the **Use XPath and ESQL equivalent grammar** check box.

What to do next

Note that expressions are still checked for valid syntax appropriate in the context of the field type, but you can now use the full range of grammar supported by the runtime environment.

Using dynamic terminals

You can add, rename, and remove dynamic terminals on a node in the Message Flow editor.

Before you begin

- Add a node that supports dynamic terminals; for more details, see "Adding a message flow node" on page 618 and "Message flow node terminals" on page 497.

About this task

Some message flow nodes support dynamic input or output terminals, including the Collector, Route, and DatabaseRoute nodes. When you have added a node to the flow editor, you can add, remove, or change dynamic terminals.

Procedure

- **Adding a dynamic terminal**
  a) Right-click the node and click **Add Input Terminal** or **Add Output Terminal**.
  b) Enter a name for the new terminal and click **OK**.

     The name must be unique for the terminal type. For example, if an input terminal called *In* already exists, you cannot create a dynamic input terminal with the name *In*.

     The new terminal is displayed on the node. If a node has five or more terminals, they are displayed as a terminal group. The following example shows a Route node with more than four output terminals.  To connect a particular output terminal, click the terminal group to open the Terminal Selection dialog box, or right-click the node and select **Create Connection**.
- **Renaming a dynamic terminal**
  a) Right-click the node and click **Rename Input Terminal** or **Rename Output Terminal**.

     These options are available only if you have added one or more appropriate terminals to this node.
  b) Select from the list the name of the terminal that you want to change.

     Only dynamic terminals are listed because you cannot change the name of a static terminal.
  c) Enter a new name for the terminal and click **OK**.

     Do not rename a dynamic terminal if one of the node properties is configured to use that name.
- **Removing a dynamic terminal**
  a) Right-click the node and click **Remove Input Terminal** or **Remove Output Terminal**,

     These options are available only if you have added one or more appropriate terminals to this node.
  b) Select from the list the name of the terminal that you want to remove and click **OK**.

     Only dynamic terminals are listed because you cannot remove a static terminal. Do not remove a dynamic terminal if one of the node properties is configured to use that terminal.

What to do next

When you have added dynamic terminals to a node, connect them to other nodes in the message flow; for more information, see "Connecting message flow nodes" on page 636.

Removing a message flow node

When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

Before you begin

This topic assumes that you have completed one or more of the following tasks.

- Create a message flow, or open an existing message flow, as described in "Creating a message flow" on page 574 and "Opening an existing message flow or subflow" on page 577.
- Add a node to the message flow, as described in "Adding a message flow node" on page 618.
- Add a subflow, as described in "Adding a subflow" on page 623.
- Read background information about nodes in "Message flow nodes" on page 484.

About this task

To remove a node from a message flow, complete the following steps.

Procedure

1. Open the message flow with which you want to work.
2. Highlight the node that you want to remove and click **Edit** > **Delete**.

   The node is removed from the flow. If you have created any connections between that node and any other node, those connections are also deleted when you delete the node.
3. If you delete a node in error, you can restore it by clicking **Edit** > **Undo Delete** or pressing Ctrl+Z. The node and its connections, if any, are restored.
4. If you undo the deletion, but decide that it is correct to delete the node, click **Edit** > **Redo Delete**.

Connecting message flow nodes

When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Before you begin

Complete the following tasks.

- Add a note to your message flow, as described in "Adding a message flow node" on page 618.
- Add a subflow to your message flow, as described in "Adding a subflow" on page 623.
- Read the concept topic, "Message flow connections" on page 497.

About this task

Your message flow might contain just one MQInput node, one Compute node, and one MQOutput node. Or it might involve a large number of nodes, and perhaps embedded message flows, that provide a number of paths through which a message can travel depending on its content. You might also have some error processing routines included in the flow. You might also need to control the order of processing.

You can connect a single output terminal of one node to the input terminal of more than one node (this is known as fan-out). If you do this, the same message is propagated to all target nodes, but you have no control over the order in which the subsequent paths through the message flow are executed (except with the FlowOrder node).

You can also connect the output terminal of several nodes to a single node input terminal (this is known as fan-in). Again, the messages that are received by the target node are not received in any guaranteed order.

When you have completed a connection, it is displayed as a black line, and is drawn as close as possible to a straight line between the connected terminals. This behavior might result in the connection passing across other nodes. To avoid this problem, you can add bend points to the connection.

To view metadata information for a node, subflow, or connection:

1. Switch to the Integration Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

If you define a complex message flow, you might have to create a large number of connections. The principle is the same for every connection. You create connections either by using the mouse, or by using the Terminal Selection dialog. See "Creating node connections with the mouse" on page 637 and "Creating node connections with the Terminal Selection dialog box" on page 638 for more information.

Creating node connections with the mouse

Use the mouse to connect one node to another.

Before you begin

Read the concept topic about connections.

About this task

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow with which you want to work.
3. Click the terminal from which the connection is to be made; that is, the terminal from which the message is propagated from the current node.

   For example, you can click the Failure, Out, or Catch terminal of the MQInput node. Hold the mouse pointer over each terminal to see the name of the terminal. You do not need to keep the mouse button pressed.

   Alternatively, click **Connection** on the palette, then click the node from which the connection is to be made. The Terminal Selection dialog box opens for you to choose the terminal from which to make a connection. Click **OK**. If a node has five or more input or output terminals (for example, if you have added dynamic terminals), they are displayed in a group. The following example shows a node with more than four output nodes.  To select a particular output terminal, click the grouped output terminal to open the Terminal Selection dialog box.
4. Click the input terminal of the next node in the message flow (to which the message passes for further processing).

   The connection is made when you click a valid input terminal. The connection appears as a black line between the two terminals.

Results

To view metadata information for a node, subflow, or connection:

1. Switch to the Integration Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

What to do next

Next: add a bend point, as described in "Adding a bend point" on page 640.

Creating node connections with the Terminal Selection dialog box

Use the Terminal Selection dialog box to connect one node to another.

Before you begin

Read the concept topic about connections.

About this task

1. Switch to the Integration Development perspective.
2. Open the message flow with which you want to work.
3. Click **Connection** above the node palette.
4. Click the node from which you want the connection to be made. The Terminal Selection dialog box is displayed.
5. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
6. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

Alternatively, you can make a connection in the following way:

Procedure

1. Click **Selection** above the node palette.
2. Right-click the node from which you want to make the connection and click **Create Connection**. The Terminal Selection dialog box is displayed.
3. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
4. Click the node to which to make the connection. If this node has only one input terminal, the connection is made immediately. If this node has more than one input terminal, the Terminal Selection dialog box is displayed again, listing the input terminals of the selected node. Click the correct terminal and click **OK**.

Results

To view metadata information for a node, subflow, or connection:

1. Switch to the Integration Development perspective.
2. Open a message flow.

3. In the Message Flow editor, hold the mouse pointer over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

What to do next

Next: add a bend point, as described in "Adding a bend point" on page 640.

Inserting nodes into existing message flows

You can insert a node into an existing message flow without having to delete existing connections and create connections for the new node.

Before you begin

The following instructions assume that you have created a message flow, as described in "Creating a message flow" on page 574.

About this task

In earlier versions of IBM App Connect Enterprise, to insert a new node into a message flow, you had to delete the existing connection between two nodes, insert the new node, then create new connections between the existing nodes and the new node. From WebSphere Message Broker Version 7.0 onwards, you can insert a new node in one step by dropping it onto an existing connection, as described by the following steps.

Procedure

1. In the Message Flow editor, open the message flow into which you want to insert a new node.
2. Choose the node that you want to insert from the message flow node palette.
3. Drag the node onto the connection between the two nodes where you want the new node to be added.

   When the mouse pointer is over the connection, a label is displayed saying "Add a node here". When you release the mouse button, the node is inserted in the message flow and is automatically connected to the preceding and succeeding nodes.

Removing a node connection

The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Before you begin

Complete the following tasks:

- "Connecting message flow nodes" on page 636.
- Read the concept topic, "Message flow connections" on page 497.

About this task

To remove a connection that you have created between two nodes:

Procedure

1. Open the message flow that you want to work with.
2. Click **Selection** above the node palette.

3. Select the connection that you want to delete.

   When you hold the mouse pointer over the connection, the editor highlights the connection that you have selected by thickening its line, adding an arrowhead at the target terminal end, and annotating the connection with the name of the two terminals connected, for example Out->In.

   When you select the connection, the editor appends a small black square at each end and at every bend point of the connection, and a small arrowhead at the target terminal end. The annotation disappears when you select the connection.
4. Check that the selected connection is the one that you want to delete.
5. Right-click the connection and click **Delete**, press the Delete key, or click **Edit** > **Delete**.

   If you want to delete further connections, repeat these actions from step "3" on page 640.
6. If you delete a connection in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**.

   The connection is restored.
7. If you undo the delete, but decide that it is the correct delete action, you can right-click in the editor view and click **Redo Delete**.

   You can also delete a connection by selecting it in the Outline view and pressing the Delete key.

Results

If you delete a node, its connections are automatically removed; you do not have to do this as a separate task.

Adding a bend point

When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow. To help you to display the message flow nodes and their connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

Before you begin

- Connect the nodes
- Read the concept topic about bend points

About this task

To add a bend point:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection to which you want to add a bend point.

   The editor appends a small black square to each end of the connection to highlight it.

   a) Check that this is the correct connection.

      The editor also adds a small point (a handle) in the connection halfway between the in and out terminals that are joined by this connection.

5. Hold the mouse pointer over this point until the editor displays a black cross to indicate that you now have control of this bend point.
   a) Hold down the left mouse button and move your mouse to move the black cross and bend point across the editor view.
6. As you drag your mouse, the connection is updated, retaining its start and end points with a bend point at the drag point.

   You can move the bend point anywhere in the editor view to improve the layout of your message flow.
7. Release the mouse button when the connection is in the correct place.

   The editor now displays the bend point that you have created with a small square (such as those at the ends of the connection), and displays another two small points in the connection, one between your newly-created bend point and the out terminal, the other between the new bend point and the in terminal.

Results

If you want to add more than one bend point to the same connection, repeat these actions from step "4" on page 640 using the additional small points inserted into the connection.

What to do next

Next: align and arrange the nodes.

Removing a bend point

When you are working with a message flow in the editor view, you might want to simplify the display of the message flow by removing a bend point that you previously added to a connection between two nodes.

Before you begin

- Add a bend point
- Read the concept topic about bend points

About this task

To remove a bend point:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection from which you want to remove the bend point.

   The editor highlights the connection and its current bend points by thickening its line and appending a small black square to each end of the connection, and by indicating each bend point with a small black square. Check that this is the correct connection.
5. Right-click over the selected connection, if you added this bend point in the current edit session.
   a) Click **Undo Create Bend Point**.

   The editor removes the selected bend point.

   If you right-click in the editor view without a connection being selected, you can also click **Undo Create Bend Point** from the menu. However, this removes the last bend point that you created in any connection, which might not be the one that you want to remove.

6. Move the bend point to straighten the line if you added this bend point in a previous edit session, because you cannot use the undo action.

   When the line is straight, the bend point is removed automatically.

   When the bend point has been removed, the connection remains highlighted. Both ends of the connection, and any remaining bend points, remain displayed as small black squares.

   The editor also inserts small points (handles) into the connection between each bend point and between each terminal and its adjacent bend point, which you can use to add more bend points.
7. If you want to remove another bend point from the same connection, repeat these actions from step "4" on page 641.

Aligning and arranging nodes

When you are working in the Message Flow editor, you can decide how your nodes are aligned in the editor view.

Before you begin

This option is closely linked to the way in which your nodes are arranged. The default for both alignment and arrangement is left to right, which means that the in terminal of a node appears on its left edge, and its out terminals appear on its right edge. You can also change this characteristic of a node by rotating the icon display to right to left, top to bottom, and bottom to top.

To complete this task, you must have completed the following task:

- "Adding a message flow node" on page 618

About this task

To modify the way in which nodes and connections are displayed in the editor:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Right-click in the editor window and select **Manhattan Layout** if you want the connections between the nodes to be displayed in Manhattan style; that is with horizontal and vertical lines joined at right angles.
5. If you want to change the layout of the complete message flow:
   a) Right-click in the editor view and click **Layout**.

      The default for the alignment is left to right, such that your message flow starts (with an input node) on the left and control passes to the right.

   b) From the four further options displayed, **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**, click the option that you want for this message flow.

      The message flow display is updated to reflect your choice. As a result of the change in alignment, all the nodes in the message flow are also realigned.

      For example, if you have changed from a left to right display (the default) to a right to left display, each node in the flow has now also changed to right to left (that is, the in terminal now appears on the right edge, the out terminals appear on the left edge).

6. You might want to arrange an individual node in a different direction from that in which the remaining nodes are arranged in the message flow:
   a) Right-click the node that you want to change and click **Rotate**.
      Four further options are displayed: **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**. The option that represents the current arrangement of the node is not available for selection.
   b) Click the option that you want for this node.

Results

If you change the alignment of the message flow, or the arrangement of an individual node, or both, these settings are saved when you save the message flow. They are applied when another user accesses this same message flow, either through a shared repository or through shared files or import and export. When you reopen the message flow, you see these changed characteristics. The alignment and arrangement that you have selected for this message flow have no effect on the alignment and arrangement of any other message flow.

You can also access the editor toolbar to select other options related to the display and arrangement of nodes, for example, snap to grid. These options are defined in Message Flow editor.

Adding annotations to a message flow or node

You can add annotations to a message flow, a node, or multiple nodes. You can use these annotations to record reminders, issues arising during the development of a message flow, or informal documentation to facilitate team development.

Before you begin

The following instructions assume that you have created a message flow, as described in “Creating a message flow” on page 574.

About this task

Procedure

- **Adding an annotation to a message flow**
  a) Open the message flow that you want to annotate in the Message Flow editor.
  b) In the message flow node palette, click **Note**, then click the canvas of the Message Flow editor.
     A blank note, in editing mode, is added to the canvas.
  c) Enter your comments to the note.
  d) To exit editing mode, click the canvas outside the note.
     The note resizes according to the amount of text that you enter.
  e) To save the contents of the note, save the message flow.

- **Associating an annotation with a node**

  a) Add a note to the canvas, as described in the preceding steps.

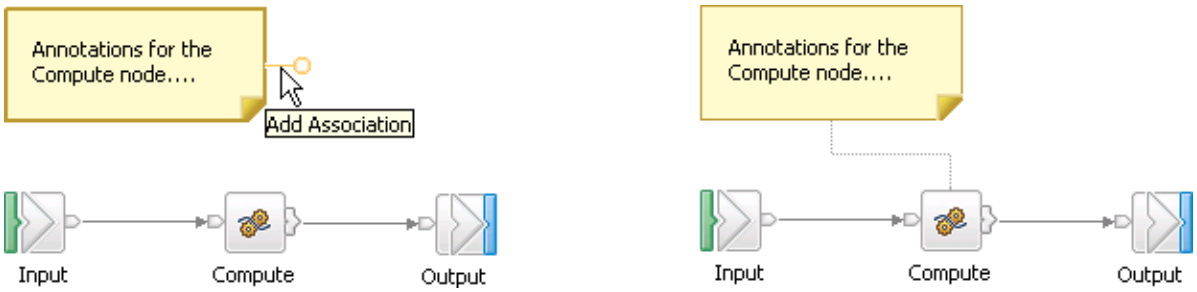
  b) To associate a note with a node in the message flow, move the mouse pointer over the edge of the note until a connector appears.

  c) Click the connector, then click the node with which you want to associate the note.

  The sticky note is associated with the selected node. This association is shown in the Message Flow editor by a dotted line.

  You can repeat these steps to associate an annotation with more than one node.

  The following diagram shows that a connector appears when you move the mouse cursor over the edge of a note, and that a dotted line represents the association between a node and a note.



What to do next

After you have added an annotation to a message flow or node, you can modify the annotation in the following ways:

- “Editing annotations on a message flow or node” on page 644
- “Copying annotations on a message flow or node” on page 645
- “Showing and hiding annotations on a message flow or node” on page 646
- “Deleting annotations from a message flow or node” on page 646

Editing annotations on a message flow or node

After you have added an annotation to a message flow or node, in the form of a sticky note, you can edit the contents of that note.

Before you begin

The following instructions assume that you have added an annotation to a message flow or node, as described in “Adding annotations to a message flow or node” on page 643.

About this task

You can add annotations to message flows or nodes to create reminders, or record details about message flow development. For example, you might want to add a list of modifications that need to be made to a message flow. As you complete the modifications, you might want to edit the annotation to reflect progress. The following steps describe how to edit an annotation.

Procedure

- **Editing the content of a note**
  a) In the Message Flow editor, open the message flow that contains the note that you want to edit.
  b) To put the note into editing mode, click the text of the note.
     The note changes color, acquires a blue border, and a cursor appears.
  c) Edit the content of the note.
     You can use standard cut, copy, and paste operations.
  d) To exit editing mode, click the canvas outside the note.
     The note resizes according to the amount of text that you enter.
  e) To save the contents of the note, save the message flow.
- **Moving a sticky note**
  – To move a sticky note, click inside the note (not on the text) so that black dots appear at each corner. You can now drag the note to a new position on the canvas. If the note is associated with one or more nodes, the association line also moves.
  – You can also move a sticky note by using cut and paste operations. Right-click the note, click **Cut**, right-click the canvas where you want the note to appear, and click **Paste**.
- **Changing the association of a sticky note**
  – To associate a sticky note with a different node, move the mouse pointer over the old node, click the end of the association line, hold down the mouse button, then drag the association line to the new node. When you release the mouse button, the sticky note is associated with the new node, as indicated by a dotted line.
  – To associate a node with a different sticky note, move the mouse pointer over the old sticky note, click the end of the association line, hold down the mouse button, then drag the association line to the new sticky note. When you release the mouse button, the node is associated with the new sticky note, as indicated by a dotted line.

Copying annotations on a message flow or node

You can copy annotations that you have added to a message flow or node to the same instance of the Message Flow editor, or to a different instance of the editor.

Before you begin

The following instructions assume that you have added an annotation to a message flow or node, as described in “Adding annotations to a message flow or node” on page 643.

Procedure

- In the Message Flow editor, open the message flow that contains the annotation that you want to copy.
  A copy of the note appears in the new location. If the original note is associated with one or more message flow nodes, those associations are not copied. You can also cut and paste a note in the same way.
- Right-click the note that you want to copy, then click **Copy**.
- Optional: If you want to copy annotations to a different Message Flow editor, switch to the Message Flow editor to which you want to copy the note.
- Right-click the Message Flow editor canvas where you want the copy of the note to appear, then click **Paste**.

Showing and hiding annotations on a message flow or node

After you have added an annotation to a message flow or node, you can hide it without having to delete it.

Before you begin

The following instructions assume that you have added an annotation to a message flow or node, as described in “Adding annotations to a message flow or node” on page 643.

About this task

Procedure

1. Open a message flow that contains notes.
2. Right-click a note, then click **Hide Notes**.

   All notes in the message flow are hidden.
3. To show all sticky notes, right-click the Message Flow editor canvas, then click **Show Notes**.

Deleting annotations from a message flow or node

Typically an annotation on a message flow or node contains temporary information, therefore you can delete it when you no longer need it.

Before you begin

The following instructions assume that you have added an annotation to a message flow or node, as described in “Adding annotations to a message flow or node” on page 643.

Procedure

- **Deleting a single annotation from a message flow**
  a) In the Message Flow editor, open the message flow that contains the note that you want to delete.
  b) Right-click the note that you want to delete, then click **Delete**.

     The note is deleted. If the note was associated with any nodes, those associations are also deleted.
- **Deleting multiple annotations from a message flow**
  a) In the Message Flow editor, open the message flow that contains the notes that you want to delete.
  b) Hold down Ctrl, highlight all the notes that you want to delete by clicking them (not on the text), then press **Delete**.

     The selected notes are deleted. If any of the notes are associated with nodes, those associations are also deleted.
- **Deleting an association between a note and a message flow node**
  a) In the Message Flow editor, open the message flow that contains the note association that you want to delete.
  b) Right-click the dotted association line between the note and the node, then click **Delete**.

     The association is deleted, but the note and node remain.
- **Deleting multiple associations between a note and message flow nodes**
  a) In the Message Flow editor, open the message flow that contains the note associations that you want to delete.
  b) Hold down Ctrl, select all the dotted association lines that you want to delete by clicking them, then press **Delete**.

     The selected associations are deleted, but the notes and nodes remain.

Testing and troubleshooting message flows

You can test your message flows in a development environment to identify issues before you deploy them to a production environment, and you can use Trace nodes to help you troubleshoot a message flow after it has been deployed. You can also create unit tests for your message flows and message flow nodes, and you can run these tests either during development or after the message flows have been deployed.

Before you begin

You must have a basic understanding of message flows and their representation in the IBM App Connect Enterprise Toolkit. See “Message flows overview” on page 483.

About this task

You can test and troubleshoot your message flows by using the following methods:

Use the Flow Exerciser

You can use the Flow Exerciser to deploy your message flow and then create and send messages to the flow. After the messages are processed, the paths that the messages took are highlighted. You can then view the structure and content of the logical message tree on any highlighted connection in the message flow. See “Testing your message flow by using the Flow Exerciser” on page 648. You might use the Flow Exerciser when you want to check the operation of a message flow.

Develop unit tests for your message flows

You can create and run unit tests to validate the operation of your App Connect Enterprise flows and message flow nodes. You can create and run the tests for flows and nodes that you are developing or for flows that have already been deployed. For more information, see “Developing integration tests” on page 1891.

Use the Flow debugger

You can use the Flow debugger to track messages through your message flow in real time. You can set breakpoints in a message flow, then step through the flow. While you are stepping through, you can examine and change the message variables and the variables used by ESQL code and Java code. See “Testing your message flow by using the flow debugger” on page 659. You might use the Flow debugger when you know that there is a problem with a message flow and you want to troubleshoot the problem.

Note: It is not possible to use the Flow debugger with the Flow Exerciser. When you use the Flow debugger, you must use a third-party tool or the Test Client to send a message to the flow; see #unique_955.

Enable user trace

You can enable user trace to show the history of processing that is carried out in a particular message flow. Built-in nodes write messages to user trace when they are processing work. You can use these messages to review the activity in a message flow and show information such as which message flow nodes were invoked, what code the nodes ran, and through which terminals the messages were sent. See Testing your message flow by enabling user trace.

Use Trace nodes

You can use Trace nodes to write out your own debugging information at specific points in the message flow. The debugging information is written to a file, to user trace, or to the system log. You can review the information after the message flow processes one or more messages. See “Testing your message flow by adding Trace nodes” on page 685. You might add Trace nodes to your message flow during development so that, when your flow is deployed in production, you can get additional information if you use user trace to troubleshoot your flow.

Enable exception log

You can enable the exception log to record data relating to issues and errors in your message flows. See “Enabling the exception log” on page 688.

Testing your message flow by using the Flow Exerciser

To check that a message flow or integration service is processing messages as expected, you can send messages to the flow by using the Flow Exerciser or an external client. You can then use the Flow Exerciser to show the path that each message took. You can also view the structure and content of the message assembly at any point in a message flow.

Before you begin

You must have the following components:

- A resource (integration service, stand-alone message flow, application that includes a message flow, or REST API).
- An integration node and associated integration server, or an independent integration server that is accessible from the IBM App Connect Enterprise Toolkit.
- If your message flow uses MQInput nodes to connect to a remote queue manager, you must have either an IBM MQ client or an IBM MQ server on the same machine as the IBM App Connect Enterprise Toolkit. To install IBM MQ components, see the IBM MQ product documentation: https://www.ibm.com/docs/en/ibm-mq.

About this task

The Flow Exerciser is a tool that is available from the flow editor and the integration service editor, but not from the REST API Editor. If you want to use the Flow Exerciser with a REST API, you must locate and open the main message flow of the REST API.

You can use the Flow Exerciser to complete the following set of tasks:

1. Create and start an integration server to use for testing your message flow.
2. Deploy the resource to an integration server and set the message flow to Record mode.
3. a. Create and send an input message to the input node of the message flow.
   b. Send a previously saved recorded message to the input node of the message flow.

   For more information, see Messages that you can use.

   You can also send messages to the message flow by using an external client.
4. Highlight and view the message path on the message flow and any subflows that are associated with the message flow.
5. View the recorded message assembly for a message that passed through a connection in the message flow.
6. Save the content of the recorded message assembly as a recorded message that you can send to the message flow later.
7. Save the content of all message assemblies in the message path as recorded messages that you can send to the message flow later.

The following videos show examples of how to use the Flow Exerciser:

- Understanding a message assembly shows how to use the Flow Exerciser to record and save a message assembly to be used as an input message to the flow.
- Creating an input message shows how to use the Flow Exerciser by creating an input message to send to the flow.
- Using an external client shows how to use the Flow Exerciser by using an external client to send a message to the flow, and how to save the message for future use.

Procedure

To check that a message flow or integration service is processing messages as expected, complete the following steps:

1. In the IBM App Connect Enterprise Toolkit, open the message flow by completing one of the following steps:
   - If the message flow is a stand-alone message flow or a message flow that is part of an application, open the flow with the flow editor.
   - If the message flow is part of an integration service, open the integration service description with the service editor.
   - If the message flow is part of a REST API, use the message flow editor to open the main message flow of the REST API, which is located under **Resources**.
2. Create or start an integration server. In the editor, click the **Start Flow Exerciser** icon () in the Flow Exerciser toolbar so that the flow recording can start.

   Before you send a message to the message flow, you must have an integration server that is connected to the IBM App Connect Enterprise Toolkit, and the integration server must be started. If you do not have a started integration server connected to the IBM App Connect Enterprise Toolkit, a message notifies you that an integration server is required. Click **OK** to use the wizard that helps you create and start an integration server. For more information, see "Creating and starting a local, independent integration server by using the Toolkit" on page 170.

   Alternatively, you can click **Cancel** and then create and start an integration server by using one of the methods described in "Creating an integration server" on page 168.

   When the integration server has been created and started, it is available to use for testing the message flow.
3. Deploy the message flow. When you have one or more started integration servers that are connected to the IBM App Connect Enterprise Toolkit, click the **Start Flow Exerciser** icon () again. The flow recording starts.

   If you have more than one started integration server that is connected to the IBM App Connect Enterprise Toolkit, you are prompted to select the integration server where you want to deploy the resource. If the message flow receives messages on an MQInput node that is configured to use a local queue manager, you can use the Flow Exerciser to send messages to that MQInput node. However, you can deploy the flow only to an integration server that is managed by a local integration node. Select the integration server where you want to deploy the message flow and click **Finish**.

   A window is presented where you are advised that a message flow cannot be deployed individually, and the owning application or library is deployed instead. Click **OK** to deploy the owning application or library to your selected integration server.

   If the resource was previously deployed to the integration server, a window is presented where you are advised that the resource is already deployed. Click **Yes** to confirm that you want to redeploy the resource or click **No** to skip the redeployment step.

   The resource is deployed to the integration server. A window is presented where you are advised that the message flow is ready to record messages. Click **OK** to close the window and return to the message flow editor. The flow editor is now surrounded by a blue border, and the text "Recording" appears. The red **Record** button is now replaced with a blue **Return flow to edit mode** button, and the message flow is set to Record mode. You cannot edit the message flow when it is in Record mode. If you want to stop the recording and return the flow to edit mode, press the blue **Return flow to edit mode** button.
4. Send messages to the message flow by using one of the following options:
   - If you are using an integration service, or if your message flow contains MQInput, HTTPInput, or SOAPInput nodes, click the **Send message** icon () in the Flow Exerciser toolbar. You can then use the **Send Message** dialog to create an input message (or select an existing input message or recorded message) and send it to the flow. For more information about creating messages or

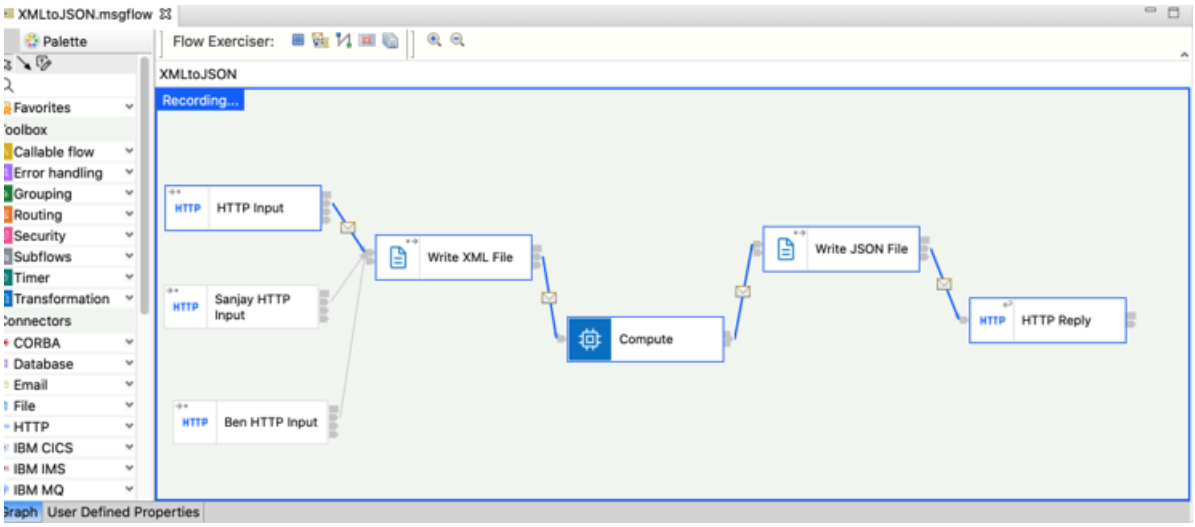
using recorded messages, see Creating input messages and Using recorded messages. The **Send message** function is not available for flows in a REST API.

- If your message flow uses input nodes that the Flow Exerciser cannot send messages to directly, use an external tool or client to form and send one or more input messages to the flow.

Click **Close** to close to the Send Message dialog.

5. View the path through the flow by using one of the following methods:
   - After you click **Close** to close to the Send Message dialog, the message paths are automatically highlighted in blue on the flow.
   - In an integration service, click the operation name in the **Integration Service Description** page to see the message paths highlighted on the subflow.
   - If your message flow uses input nodes that the Flow Exerciser cannot send messages to directly, and you used an external tool or client to form and send one or more input messages to the flow,

you must click the **View path** icon ( ) in the Flow Exerciser toolbar to highlight message paths on the flow.



The diagram shows a flow where the message paths are highlighted. At least one message passed through each highlighted connection.

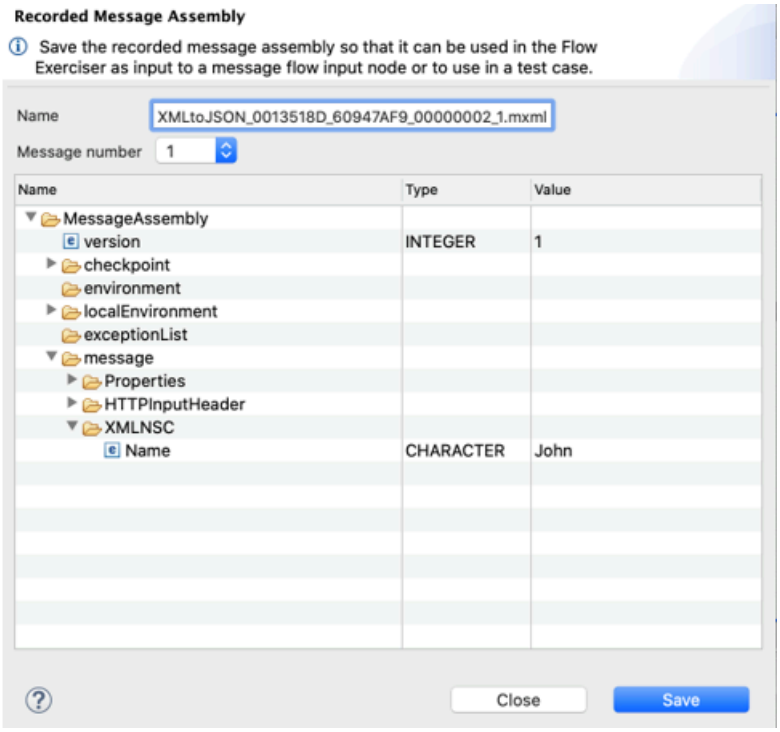
- The highlighted connections do not distinguish between the paths that are taken by different messages. If you send more than one message to the flow, you must inspect each highlighted connection to see which messages passed through that connection.
- If you send a single message to the flow and the message passes through a connection multiple times, the logical message tree is captured as a separate message instance each time the message passes through a connection.
- By default, a maximum of 200 message instances are displayed in the message flow, but you can change this value in the preferences (**Windows** > **Preferences** > **Integration Development** > **Flow Exerciser**).
- If the number of message instances that are captured exceeds the number that is configured in the preferences, you are prompted to choose whether to view the configured number of recorded messages, or whether to view all the recorded messages:
  - If you opt to view the configured maximum number of recorded messages, you might not see whole sequences of messages.
  - If you opt to view all the messages, performance can be impacted.

When you finish viewing the path through the flow, choose one of the following options:

- Click **Clear Recorded Messages** to clear the recorded messages. The blue line that highlights the path taken through the flow is no longer shown.
- Click **Return flow to Edit Mode** to clear the recorded messages and return the message flow to Edit mode. The blue line that highlights the path taken through the flow is no longer shown. You can then edit the message flow and redeploy the flow to the integration server, as described in step "3" on page 649. You can then resend the message, as described in step "4" on page 649.
- Leave the message flow in Record mode, and view the message assembly, as described in step "6" on page 652.

6. Optional: When the message flow is in Record mode, and the blue line that highlights the path taken through the flow is visible in the Flow Exerciser, you can view the recorded message assembly of the message.
   The message assembly is a logical representation of the message flow that is being handled at a particular point in the process. For more information, see Message Assembly editor.

   Open a read-only view of the message assembly by selecting a message icon that is on a connection between any two nodes in the path of the message flow.



   The view shows the logical message tree at that point in the flow. The logical message tree is referred to as the *message assembly*. The message assembly consists of four trees:

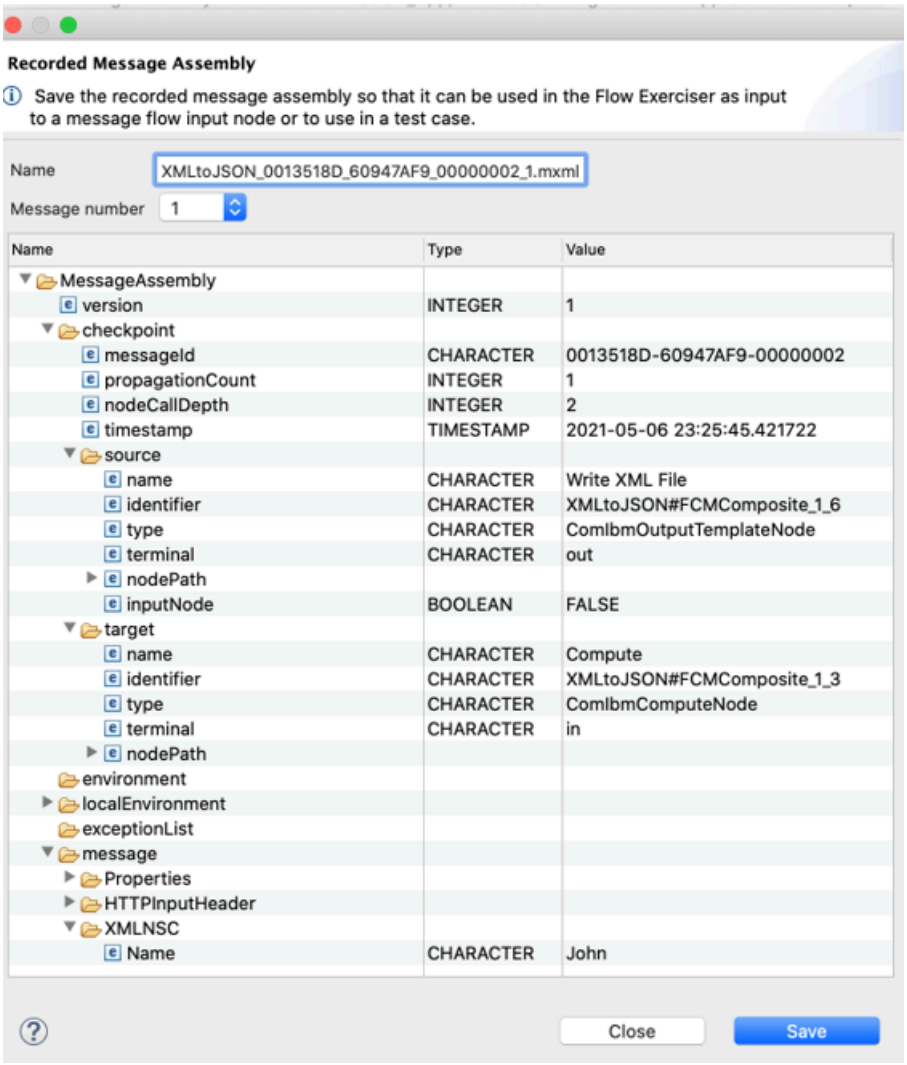
   - *environment*
   - *localEnvironment*
   - *exceptionList*
   - *message*

   The message tree contains message headers and the message body. The message body part of the message assembly is automatically expanded when the view is shown.

   A checkpoint folder is also shown in the message assembly. The checkpoint shows details of where in the flow the message was recorded. It shows the source node and the target node details.

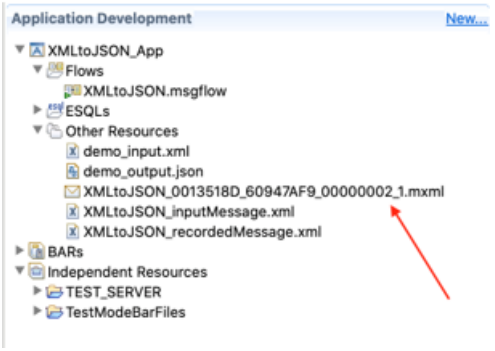
   If you send multiple messages to the flow, you can select which message you want to view by selecting the appropriate entry in the **Message number** drop-down menu. When you change the

selection in the **Message number** menu, the name of the message is automatically updated in the **Name** field .



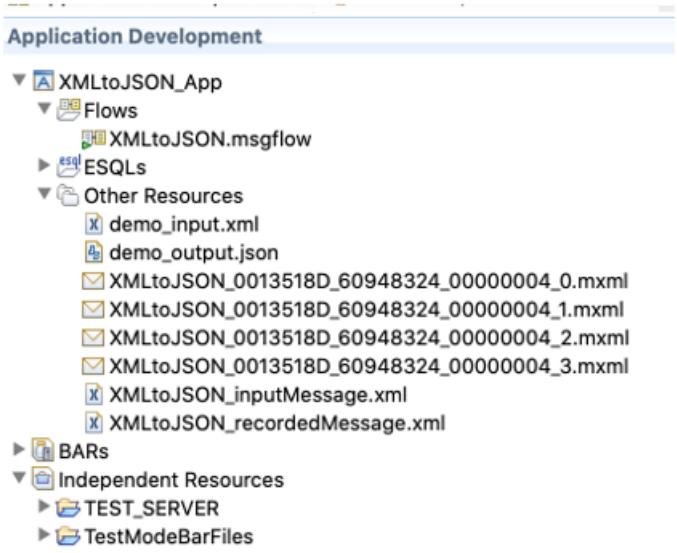
If you want to save the message assembly, follow the instructions in step “7” on page 654.

7. Optional: When you view the message assembly in the read-only view, you can save it by completing the following steps:
   a) Optional: You can change the name of the message assembly by overwriting the default name that appears in the Name field of the **Recorded Message Assembly** window. The name of the message assembly must have the extension `.mxml`.
   b) Click **Save**.



   - The message assembly is saved under Other Resources in the application that contains the message flow that is being recorded.
   - You can now view a different message number in the same window and save that message too.
   - When you have finished viewing and saving the message assembly, select **Close** to close the window. You can now see the saved message assembly under the application in the Application Development view.
8. Optional: Click **Save All** if you want to save the message assemblies from all the connections in the highlighted path in the Flow Exerciser.

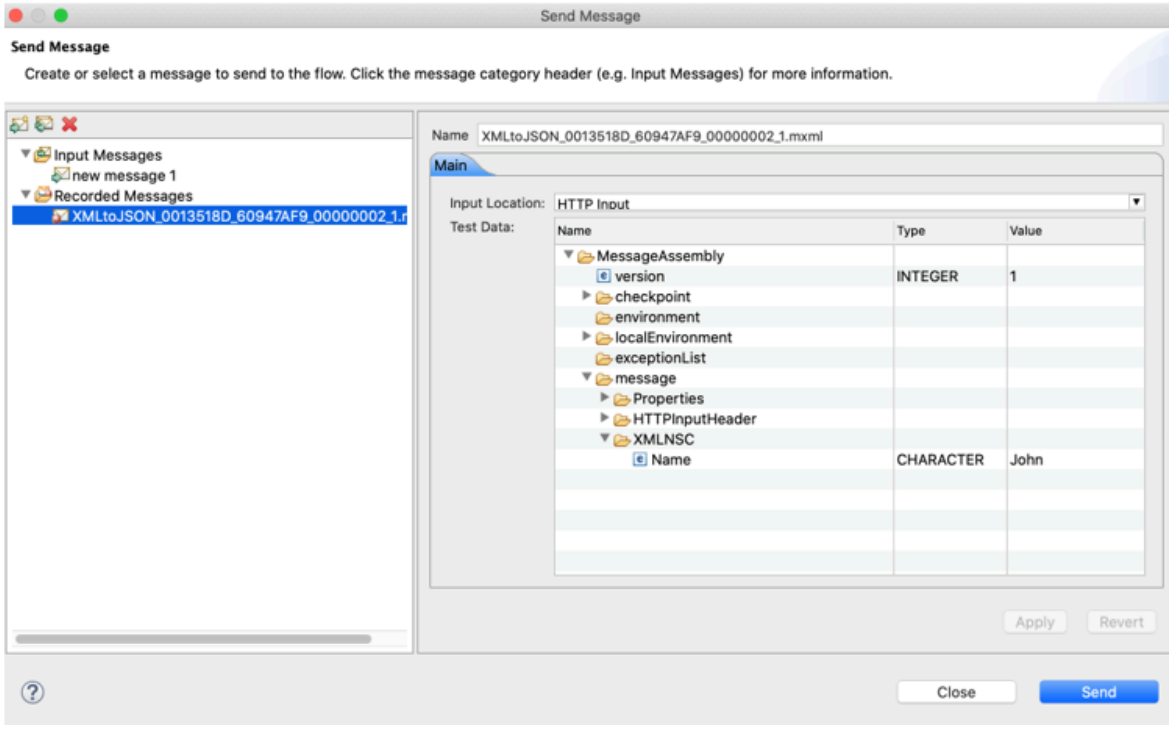
   All the message assemblies for every invocation of the flow now appear in the Application Development View under the application that contains the message flow that was recorded.



   These message assemblies are also visible under Recorded Messages the next time that you select **Send Message**.

   These message assemblies can also be used as resources for integration tests for your message flow nodes.

9. Optional: Edit a message assembly by completing the following steps:
   a) Double-click the saved message assembly under the Application Development view in the IBM App Connect Enterprise Toolkit.
   b) Select **Open with** > **Message Assembly Editor** from the menus.
      The message assembly opens in the Message Assembly Editor.
   c) Edit the values under the Name, Type, and Value columns.
   d) Save your changes.
   e) Edit the message flow to expect the updated values. For example, if your flow has a Compute node, update the ESQL. For more information, see Editors.
10. If you save a recorded message assembly, you can send it as a message to a message flow, by completing the following steps:
   a) Click **Send Message**.
      - The saved, recorded message assembly is shown under Recorded Messages.
      - A read-only view of the contents of the recorded message assembly is shown under the Main tab.
      - If you need to change the message assembly, select it from the Application Development view to open it, then edit it in the Message Assembly Editor.
      - The input node is not saved with the message assembly, and you can select which input node to send it to. All input nodes that are in the message flow are listed in the **Input Location** field.
   b) Press **Send**.



The message is sent and the path of the message is highlighted in blue in the flow editor, as shown in step “5” on page 650.

Messages that you can use with the Flow Exerciser to check that a message flow processes messages as expected

You can create input messages and save recorded messages. You can then use either of these types of message to check that a message flow is processing messages as expected.

Input messages

Input messages are messages that are processed by the input node in a message flow.

Input messages can be generated and sent from a third-party tool or client, or created by using the Flow Exerciser. If you create input messages by using the Flow Exerciser, you can create messages for message flows that contain the following input nodes:

- MQInput node
- HTTPInput node
- SOAPInput node

If the message flow is part of a REST API, you cannot use input messages created by the Flow Exerciser; however, you can use an external tool or client to form and send the input messages to the flow.

If the input node in the message flow expects an XML message from an associated message model, the message structure is provided and can be edited to produce an appropriate sample input message. Alternatively, you can manually create a new input message, or import an existing message from your file system.

You can duplicate, edit, and delete the messages to create a set of input messages that validate the operation of your message flow, and you can export the content of an input message to a file. To use input messages, you must have a method of invoking the flow.

For more information, see Creating input messages.

Recorded messages

Recorded messages are messages that are captured when a message flow is in recording mode. These messages contain the logical message tree assembly that is the result of an input node processing an input message. You can save a recorded message and then use the recorded message when you do not have a method of invoking the flow, or an external system to receive the message.

Note: To create the recorded message, you must have a method of invoking the flow.

You can save and reuse recorded messages that are sent to the following input nodes:

- MQInput node
- HTTPInput node
- SOAPInput node
- FileInput node
- MQTTSubscribe node

If the message flow is part of a REST API, you cannot use recorded messages; however, you can use an external tool or client to form and send input messages to the flow.

You save recorded messages by clicking the highlighted connection from the input node to the next node in the message flow, and clicking the **Save** icon. You can rename a recorded message but you cannot edit the content of a recorded message, or create a recorded message from scratch.

When you send a recorded message to the message flow, the message is sent to the output terminal of the input node of the message flow. A recorded message is not processed by the input node, and so you do not need a method of invoking the flow.

HTTPReply and SOAPReply nodes can detect that the message that is being processed by the flow is a recorded message. If one of these nodes receives a recorded message, the node suppresses the reply.

(Attempting to send a reply would generate an error because there is no external system to receive the reply.)

For more information, see Using recorded messages.

If you create a set of input messages and recorded messages, these messages can be included with the message flow if you export the message flow (or application that contains the message flow) as a project interchange file. However, you cannot copy the messages to another message flow.

Creating input messages by using the Flow Exerciser to check that your message flow processes messages as expected

You can create a new input message, or duplicate or edit a previously saved input message by using the Flow Exerciser. You can then send the message to the flow.

Before you begin

You must have a message flow that is in recording mode and that contains MQInput, HTTPInput, or SOAPInput nodes.

You cannot use the Flow Exerciser to send messages to an MQInput node in a flow in the following situations:

- The MQInput node is configured with a policy.
- The MQInput node has the `MQ connection` property set to `Client channel definition table (CCDT) file`.
- The message flow is deployed to a remote integration node, and the message flow receives messages on an MQInput node that is configured to use a local queue manager.

Instead, you can use an external tool or client to send messages to the message flow.

If the message flow is part of a REST API, you cannot use recorded messages or input messages created by the Flow Exerciser; however, you can use an external tool or client to form and send input messages to the flow.

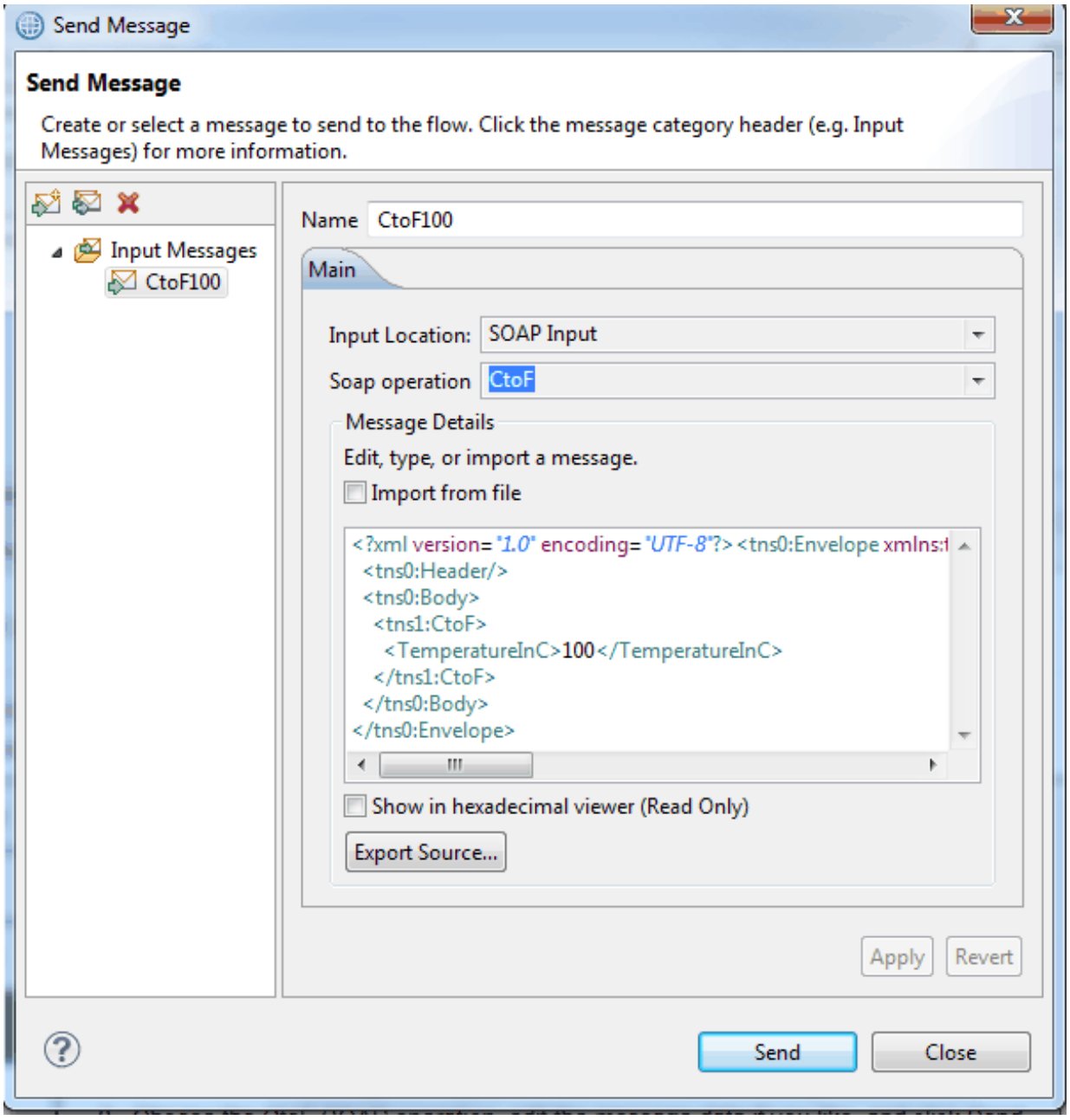
Procedure

1. Click the **Send message to the flow** icon () in the Flow Exerciser toolbar.

   The **Send Message** dialog opens.

2. Click the **New message** icon ( )

   A default message is displayed. If the input node in the message flow expects an XML message from an associated message model, the message structure is provided. If the input node does not have an associated message model, the message content is blank.



3. Enter a name for your message or accept the default value.
4. Optional: If you are using an integration service that contains more than one operation, click the **Soap operation** field and select the operation name.
5. Finalize the message by completing one of the following steps:
   - If the message structure is provided, you can replace the default values with appropriate values of your own, or you can leave the message as it is.
   - You can manually create the message structure and values from scratch.
   - You can click **Import** and import an XML message from a file.
6. Optional: Click **Apply** to save any changes.

7. Click **Send** to send the message to the message flow.

   The message is sent to the flow, and the connections through which the message passed are highlighted.

What to do next

In the Flow Exerciser, you can create additional input messages by duplicating and editing existing input messages, and you can export the structure and content of an input message to a file.

Using recorded messages with the Flow Exerciser to check that your message flow processes messages as expected

From the **Send Message** dialog, you can select a previously saved recorded message and send it to the message flow.

Before you begin

You must have the following components:

- A message flow that is in recording mode
- One or more recorded messages

About this task

Recorded messages are created by sending input messages to a message flow by using the Flow Exerciser. A recorded message contains the logical message tree assembly that is the result of an input node processing an input message. You cannot modify the recorded message or create one from scratch. For more information, see Messages that you can use.

Procedure

1. Click the **Send message to the flow** icon () in the Flow Exerciser toolbar.

   The **Send Message** dialog opens. The recorded messages that you previously saved are listed under **Recorded Messages** in the navigator.
2. Select a recorded message to see the content.

   You can change the name of the recorded message if you want, by typing a new name and clicking **Apply**. However, you cannot duplicate the message or edit the content of the message.
3. By default, the recorded message is sent to the input node where it was saved. If there are other input nodes of the same type in the message flow, you can select one of these other input nodes instead.

   Note: If you select an input node other than the input node where the message was saved, you might get an error in your flow when you send the message. For example, the selected input node might send messages in a different format.
4. Click **Send**.

   The message is sent to the flow, and the connections through which the message passed are highlighted.

Testing your message flow by using the flow debugger

Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Before you begin

If you are new to debugging, see: “Flow debugger overview” on page 660.

Deploy your message flow to an integration server in an integration node and make sure that the integration node is running. See: “Deployment rules and guidelines” on page 2465.

About this task

To debug a message flow, perform the following tasks. You might want to vary the tasks that you perform and repeat certain tasks, depending on your particular debugging requirements. If your message flow contains WebSphere Adapters nodes, also see "Debugging message flows that contain WebSphere Adapters nodes" on page 683.

Procedure

1. Start the flow debugger.

   Set the required preferences, then start debugging by attaching the flow debugger to an integration server. You can then send test messages along the flow. See: "Starting the Flow debugger" on page 665.

   Note: It is not possible to use the Flow debugger with the Flow Exerciser. When you use the Flow debugger, you must use a third-party tool or the Test Client to send a message to the flow; see #unique_955. If your integration server is in recording mode, you must stop recording mode before you can launch the Flow debugger.
2. Work with breakpoints.

   Add and manipulate breakpoints in your message flow. See: "Working with breakpoints in the flow debugger" on page 669.
3. Follow the progress of a test message.

   Use breakpoints to pause the progress of a test message so that you can observe its behavior. See: "Stepping through message flow instances in the debugger" on page 672.
4. View message data.

   View (and change) data in messages, ESQL code, or Java code as debugging progresses. See: "Debugging data" on page 677.
5. Manage message flows.

   During a debugging session, there are various administrative tasks you might need to do. When you have finished debugging, detach the debugger from the integration server. See: "Managing flows and flow instances during debugging" on page 680.

Flow debugger overview

Use the Flow debugger in the IBM App Connect Enterprise Toolkit to track messages through your message flows.

Use the Debug perspective in the IBM App Connect Enterprise Toolkit to use the Flow debugger. For an introduction to the Debug perspective and the views it presents, see #unique_971.

You can set breakpoints in a message flow, then step through the flow. While you are stepping through, you can examine and change the message variables and the variables used by ESQL code and Java code. You can debug a wide variety of error conditions in flows, for example:

- Nodes that are wired incorrectly (for example, outputs that are connected to the wrong inputs)
- Incorrect conditional branching in transition conditions
- Unintended infinite loops in flow

From a single IBM App Connect Enterprise Toolkit, you can attach the Flow debugger to one or more integration servers, and debug multiple flows in different integration servers (and therefore multiple messages) at the same time. However, an integration server can be debugged by only one user at a time. Therefore, if you attach your Flow debugger to an integration server, another user cannot attach a Flow debugger to that same integration server until you have ended your debugging session.

When you debug message flows, use an integration node that is not being used in a production environment. Debugging might degrade the performance of all message flows in the same integration server and message flows in other integration servers that share the same integration node because they might be affected by resource contention.

Debugging code in message flow nodes

You can use the Flow debugger to examine the behavior of code in message flow nodes.

After you have deployed a message flow, you can set a breakpoint just before one of the nodes listed in this section so that, when the flow pauses at the breakpoint, you can step through the code line by line. This allows you to examine the logic, and check the actions taken and their results. You can set additional breakpoints and you can also examine and change variables.

The following nodes can contain ESQL code modules:

- Compute node
- Filter node
- Database node

The following nodes can contain Java code modules:

- User-defined nodes
- JavaCompute node

Restrictions

The following restrictions apply when you debug a message flow:

- You cannot use the Flow debugger with the Flow Exerciser. When you use the Flow debugger, you must use a third-party tool or the Test Client to send a message to the flow; see #unique_955.
- You must use the same version of the integration node and the IBM App Connect Enterprise Toolkit; for example, you cannot use the IBM App Connect Enterprise Toolkit Version 12.0 to debug a message flow that you have deployed to an integration node at an earlier version.
- You should not debug message flows over the Internet, because there might be security issues.
- You can debug a message flow that contains a Mapping node, but you cannot step into the graphical data mapping to view or set breakpoints on the transforms.

The following topics provide reference information to help you use the Flow debugger effectively:

- “Flow debugger shortcuts” on page 661
- “Flow debugger icons and symbols” on page 662

You can also use the “Java Debugger” on page 664 provided by the Java Development tools to debug Java code within the IBM App Connect Enterprise Toolkit.

Flow debugger shortcuts

You can use function keys and shortcut keys to complete actions in the flow debugger views and windows.

Shortcut keys are shown as a pair that you press together, followed by a subsequent key, for example `Shift-F10, C` means hold the Shift key down and press F10, then release both and press key C.

The following tables describe the main shortcuts that are available in the debug session:

- “Debug view” on page 662
- “Breakpoints view” on page 662
- “Flow Breakpoint Properties dialog” on page 662
- “Variables view” on page 662

To see a complete list of all the shortcuts that are available, press `Shift-F10` and release; the contextual menu is displayed.

Debug view

Key combination	Function
Shift-F10, C	Run to completion
Shift-F10, E	Disconnect
F5 or Shift-F10, I	Step into
F8 or Shift-F10, M	Resume
Shift-F10, N	Terminate All
F6 or Shift-F10, O	Step over
Shift-F10, T	Terminate
F7 or Shift-F10, U	Step return
Shift-F10, V	Terminate and Remove

Breakpoints view

Key combination	Function
Shift-F10, A	Select All
Shift-F10, B	Add breakpoint
Shift-F10, D	Disable the selected breakpoints
Shift-F10, E	Enable the selected breakpoints
Shift-F10, L	Remove all breakpoints
Shift-F10, O	Remove the selected breakpoints

Flow Breakpoint Properties dialog

Key combination	Function
E	Enable the breakpoint
Alt-R, <space>	Restrict the breakpoint to the selected flow instances

Variables view

Key combination	Function
Shift-F10, A	Select All
Shift-F10, C	Change the value of the selected flow variable
Shift-F10, V	Copy variables

Flow debugger icons and symbols
The Debug perspective uses various debugger icons and symbols.

This topic describes the icons and symbols used in the Debug perspective and its views:

Debug perspective

These icons and symbols are used in the Debug perspective *outside* any individual view.

Icon or Symbol	Description
	Debug perspective (symbol)
	Attach to Flow runtime environment (icon)

Debug view

Icon or Symbol	Description
	Debug view (symbol)
	Flow engine (symbol)
	Flow (symbol)
	Flow instance paused (symbol)
	Flow instance running (symbol)
	Flow instance terminated (symbol)
	Stack frame (symbol)
	Detach from the selected flow engine (icon)
	Resume flow execution (icon)
	Run the flow to completion (icon)
	Step into subflow (icon)
	Step over node (icon)
	Step out of subflow (icon)
	Step into source code (icon)

Message Flow editor

These icons and symbols in the message flow editor are specific to the flow debugger.

Icon or Symbol	Description
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)

Icon or Symbol	Description
	Paused at breakpoint (symbol)
	Source code available (symbol)
	Error or exception (symbol)

Breakpoints view

Icon or Symbol	Description
	Breakpoints view (symbol)
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)
	Remove selected breakpoints (icon)
	Remove all breakpoints (icon)

Variables view

These icons and symbols in the Variables view are specific to ESQL.

Icon or Symbol	Description
	Variable view (symbol)
	Tree reference variable (symbol)
	Message (symbol)
	ESQL reference variable (symbol)
	ESQL constant (symbol)
	ESQL scalar variable (symbol)
	ESQL schema variable (symbol)
	ESQL module variable (symbol)

Java Debugger

The Java Development Tools include a debugger that enables you to detect and diagnose errors in your programs running on local or remote systems. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

For further information about the Java debugger, refer to the Java Development User Guide - Debugger. (This link works only if you are accessing this documentation from the IBM App Connect Enterprise Toolkit.)

Starting the Flow debugger

To start the Flow debugger, you must attach it to an integration server. When the Flow debugger is started, you can introduce test messages to your message flow.

About this task

Note: It is not possible to use the Flow debugger with the Flow Exerciser. If your integration server is in recording mode, you must stop recording mode before you can launch the Flow debugger.

Complete the following tasks to start the debugger:

Procedure

1. "Attaching the flow debugger to an integration server for debugging" on page 665
2. Optional: "Debug: putting a test message on an input queue" on page 666
3. Optional: "Debug: getting a test message from an output queue" on page 668

Attaching the flow debugger to an integration server for debugging

Before you can debug your message flow, you must attach the flow debugger to the integration server where your flow is deployed, then start a debugging session.

Before you begin

- If authorization permissions are required, see: "Authorizing users for administration" on page 2516
- Create a message flow. See: Chapter 6, "Developing integration solutions," on page 481
- Deploy your message flow to an integration server. See: "Deployment rules and guidelines" on page 2465
- Start the integration node. See: "Starting and stopping an integration node" on page 247

About this task

From the IBM App Connect Enterprise Toolkit, you can attach the flow debugger to multiple integration servers that are running on the same or on different host computers, and debug their flows (and therefore multiple messages) simultaneously.

An integration server can be debugged by only one user at a time. If you attach your debugger to an integration server, another user cannot attach a debugger to that same integration server until you have ended your debugging session.

Procedure

To attach the debugger to an integration server:

1. Set the JVM debug port by modifying the `server.conf.yaml` configuration file for your integration server.
   a) Open the `server.conf.yaml` configuration file by using a YAML editor.

      If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid characters in YAML files and would cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   b) In the Resource Managers section of the `.yaml` file, specify a value for the **`jvmDebugPort`** property, which sets the JVM debug port to be used for debugging flows in the Toolkit.

2. Restart the integration server for the changes to take effect.
3. In the IBM App Connect Enterprise Toolkit, click the down-arrow to the right of the Debug button in the action bar, and then click the **Debug configurations** menu item.
4. In the **Debug configurations** dialog, right-click **IBM App Connect Enterprise debug**, and then click **New**.
5. Enter a name for the new debug configuration in the **Name** field; if you do not specify a name, it will be called `New_configuration [IBM App Connect Enterprise Debug]` by default. In the **Connect** tab, set the host name and the Java debug port to match the details of your integration server, as specified in the **`jvmDebugPort`** property in the `server.conf.yaml` file. Click **Debug**.
6. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Application Development view.
7. Add a breakpoint to a connection that leads out of the input node to ensure that the message flow does not run to completion before you can begin to debug it.

   The breakpoint appears as  . For information about adding a breakpoint, see "Working with breakpoints in the flow debugger" on page 669.
8. Switch to the Debug perspective.
9. Optional: If the message flow that you are going to debug references additional resources in other projects, such as ESQL in libraries or independent projects, or Java in Java projects, add those resources by completing the following steps:
   a) In the **Debug** view, right-click the debug configuration that you created in step 5 and then click **Edit Source Lookup** from the pop-up menu. If you did not specify a name when you created the debug configuration, it will have the default name `New_configuration [IBM App Connect Enterprise Debug]`.

      You can use **Edit Source Lookup Path** to tell the debugger where to look for your source files for message flows and related resources such as ESQL and Java during debugging.
   b) Click **Add**, and select the type of source to add to the lookup path.

      The lookup path can be an Eclipse project name, an external folder, or a compressed (`.zip`) file. You can specify multiple locations, but the debugger always looks first in the message flow project that you specify in the **Edit Source Lookup Path** dialog.
   c) Select the resources to include in the lookup path, and click **OK**.
   d) Click **Add** to include more resources in the lookup path, click **Up**, or **Down** to modify the order of the resources.
   e) Click **OK** to exit the **Edit Source Lookup Path** dialog, and save your changes.
10. When the next message comes into your flow and arrives at a breakpoint that you added after the input node, the flow pauses, the breakpoint icon is highlighted:  , and you can start debugging.
11. In the **Debug view**, double-click the message flow that you want to debug. The message flow opens in the Message Flow editor. You can now add more breakpoints, start stepping over the flow, and so on.

What to do next

Send a message through your message flow.

Debug: putting a test message on an input queue
You can put a message on an input queue to test a message flow that you are debugging.

Before you begin

Complete the steps described in "Attaching the flow debugger to an integration server for debugging" on page 665.

About this task

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node.

You can use the command line interfaces or IBM MQ Explorer to put a message to a queue.

Note: It is not possible to use the Flow debugger with the Flow Exerciser.

You can also use the Test Client as a repeatable alternative. To use the Test Client, complete the steps described in the following sections.

Note: The Test Client is not enabled by default. To use the Test Client, you must enable this facility in the Test Client preferences; see preferences.

- "Using enqueue in the Test Client" on page 667
- "Adding data to your message" on page 667
- "Optional: Using a file of sample data" on page 668

If the message is processed by the message flow and is put on an output queue, you can retrieve it from that queue. See: "Debug: getting a test message from an output queue" on page 668.

Using enqueue in the Test Client

About this task

To configure an enqueue event in the Test Client so that you can use it to send a test message:

Procedure

1. Switch to the Integration Development perspective.
2. Open your message flow in the Message Flow editor.
3. Right-click the input node of the message flow and click **Test**.

   The message flow is deployed and the Test Client editor opens and displays the **Events** window.
4. On the toolbar of the Test Client editor, under **Message Flow Test Events**, click the **Enqueue** icon  .
5. Under **Detailed Properties**, enter the names of the queue manager and the queue for the input node for this flow.

   Queue manager names are case-sensitive; make sure that you enter the name correctly.

   If you are putting a message onto an input queue that is on a remote computer, ensure that the queue manager of the associated integration node has a server-connection channel called SYSTEM.BKR.CONFIG.
6. If you are putting a message onto a remote queue, enter values to identify the host and port of the computer that is hosting the queue.
7. Optional: If you want to save the Test Client file, complete the following steps:

   a) Click **File** > **Save** and select the project where you want to save the file.

   b) Enter a name for the file and click **Finish**.

   The Test Client file is saved in the **Flow Tests** folder in the project.

Adding data to your message

About this task

If you want to add just a small amount of test data in your test message, type the data into the **Source** section of the **Message** pane:

Procedure

1. Open your Test Client file.
2. Type your test data directly into the **Source** section of the Message pane.
3. Put the test message by clicking **Send Message**.

Optional: Using a file of sample data

About this task

If you want your test message to contain a larger quantity of sample data (for example some structured XML), you can import a file containing that data into the Test Client:

Procedure

1. In the **Events** tab of your Test Client file, click **Import Source**.
2. Select the file you want to use as the content for the test message, and click **Open**.
   The contents of the selected file is added to the **Source** pane.
3. Click **File** > **Save** when you have finished.
4. Put the test message by clicking the **Send Message** button.

Debug: getting a test message from an output queue
You can get a message from an output queue to test a message flow that you are debugging.

Before you begin

Completed the following tasks:

- Chapter 6, “Developing integration solutions,” on page 481
- Chapter 7, “Deploying integration solutions,” on page 2463
- “Attaching the flow debugger to an integration server for debugging” on page 665
- “Debug: putting a test message on an input queue” on page 666

About this task

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node and retrieving it from an MQOutput node.

You can use the command line interfaces or IBM MQ Explorer to get a message from an output queue.

You can also use the Test Client as a repeatable alternative.

Note: The Test Client is not enabled by default. To use the Test Client, you must enable this facility in the Test Client preferences; see preferences.

To use the Test Client, complete the following steps:

Procedure

1. Switch to the Integration Development perspective.
2. Open your message flow in the Message Flow editor.
3. Right-click the input node of the message flow and click **Test**.
   The message flow is deployed and the Test Client editor opens and displays the **Events** window.
4. On the toolbar at the upper right of the Test Client editor, under **Message Flow Test Events**, click the **Dequeue** icon  .
5. Under **Detailed Properties**, enter the name of the queue manager and output node queue.

6. Click **Get Message** to read a message from the queue.
7. Optional: If you want to save the Test Client file, complete the following steps:
   a) Click **File** > **Save** and select the project where you want to save the file.
   b) Enter a name for the file and click **Finish**.

   The Test Client file is saved in the **Flow Tests** folder in the project.

Results

When a message is available on an output queue, you can see it in the Test Client editor.

Working with breakpoints in the flow debugger

When you have started a debugging session by attaching the debugger to an integration server, you can set breakpoints to control where the message flow will pause.

About this task

Use the following tasks to manage breakpoints:

- "Adding breakpoints in the flow debugger" on page 669
- "Restricting breakpoints in the flow debugger to specific flow instances" on page 670
- "Enabling and disabling breakpoints in the flow debugger" on page 671
- "Removing breakpoints in the flow debugger" on page 671

What to do next

After you have set one or more breakpoints in the message flow, continue your debugging session by stepping through the message flow, pausing at each active breakpoint. See: "Stepping through message flow instances in the debugger" on page 672.

You can also examine message data, code, and mappings at appropriate points. See: "Debugging data" on page 677.

Adding breakpoints in the flow debugger

Add breakpoints to connections in your message flow to control where flow processing will pause.

Before you begin

Attach the flow debugger to the integration server where your flow is deployed. See: "Attaching the flow debugger to an integration server for debugging" on page 665.

About this task

You can add breakpoints to the connections of a message flow that is open in the Message Flow editor. Each breakpoint that you add to a flow is also automatically added to all other instances of the flow and you do not need to restart any of the instances.

Every breakpoint is automatically enabled when you add it to a connection and the connection is flagged with the enabled breakpoint symbol  .

Manually set a breakpoint after the collector node or any other multithreaded node. When you use the Debug perspective on the node, you see that the thread has been ended.

To add breakpoints to the connections of a message flow:

Procedure

1. Switch to the Debug perspective.

2. Add breakpoints to the appropriate connections.

   Use any of the following methods:

Option	Method
Add breakpoints individually to selected connections.	a. In the Message Flow editor, right-click the connection where you want to set the breakpoint. b. Click **Add Breakpoint**.
Add breakpoints simultaneously to all connections *entering* a selected node.	a. In the Message Flow editor, right-click the node before which you want to set breakpoints. b. Click **Add Breakpoints Before Node**.
Add breakpoints simultaneously to all connections *leaving* a selected node.	a. In the Message Flow editor, right-click the node after which you want to set breakpoints. b. Click **Add Breakpoints After Node**.

What to do next

After you have set one or more breakpoints in the message flow, step through the flow, pausing at each active breakpoint. See: "Stepping through message flow instances in the debugger" on page 672.

You can also examine message data, code, and mappings at appropriate points. See: "Debugging data" on page 677.

Restricting breakpoints in the flow debugger to specific flow instances

Breakpoints can be applied to particular flow instances, instead of all instances, which is the default behavior.

Before you begin

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 669.

About this task

When you add a breakpoint to a message flow in the Message Flow editor, the breakpoint automatically applies to all instances of the flow. However, you can restrict a breakpoint to one or more instances of a flow. This enables you to work more easily with just those instances that you are currently interested in, rather than with all instances.

To restrict a breakpoint to one or more flow instances:

Procedure

1. Switch to the Debug perspective.
2. In the Breakpoints view, right-click the breakpoint that you want to restrict, then click **Properties** to open the **Flow Breakpoints Properties** window.
3. In the **Restrict to Selected Flow Instance(s)** list box, select those instances to which you want to restrict the breakpoint.
   - You must have at least one instance active; if not, the **Restrict to Selected Flow Instance(s)** list box is empty.
   - If any instance is currently paused at the breakpoint, all check boxes in the **Restrict to Selected Flow Instance(s)** list box are disabled and you cannot select them.

4. Click **OK**.

What to do next

Now you can add additional breakpoints (if needed), step through the flow instance, and work with data:

- "Adding breakpoints in the flow debugger" on page 669
- "Stepping through message flow instances in the debugger" on page 672
- "Debugging data" on page 677

Enabling and disabling breakpoints in the flow debugger
You can disable breakpoints that are currently enabled, and vice versa.

Before you begin

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 669.

About this task

Message flow processing pauses only at breakpoints that are enabled. By controlling which breakpoints are enabled and which are disabled, you can, for example, allow processing to continue to the part of a flow that you are interested in without having to continually add and remove breakpoints.

The following symbols identify breakpoints:

 Enabled breakpoint

 Disabled breakpoint

If you disable all the breakpoints in a message flow, you cannot perform any other debugging tasks until you add a new breakpoint, or enable an existing breakpoint.

To change the state of breakpoints:

Procedure

1. Switch to the Debug perspective.
2. In the Breakpoints view, select one or more breakpoints that you want to enable or disable.
3. Right-click the selected breakpoints and click **Enable** or **Disable**.
4. Optional: to change the state of a single breakpoint, right-click the breakpoint and click **Properties**. Select or clear the **Enabled** check box as required, then click **OK**.

Results

The state of breakpoints is changed in all instances of the message flow where they are set.

What to do next

If you have finished debugging, continue with: "Debug: ending a session" on page 683.

Removing breakpoints in the flow debugger
Remove breakpoints that are no longer required from connections in your message flow.

Before you begin

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 669.

About this task

The following symbols identify breakpoints:

- Enabled breakpoint
- Disabled breakpoint

If you remove a breakpoint from a message flow, it is automatically removed from all instances of the message flow where it is set.

If you remove all the breakpoints that you have added to your message flow, you cannot perform any other debugging tasks until you add a new breakpoint.

To remove breakpoints:

Procedure

1. Switch to the Debug perspective.
2. Remove the breakpoints.

   Use one of the following methods, depending on how many breakpoints you want to remove:

Option	Method
Remove individual breakpoints.	a. In the Message Flow editor, right-click the breakpoint that you want to remove, then click **Remove Breakpoint**.
Remove several breakpoints simultaneously.	a. Click the **Flow Breakpoints** tab to show the Breakpoints view. b. Select one or more breakpoints that you want to remove. c. Click the **Remove Selected Breakpoints** icon on the toolbar, or right-click the selected breakpoints, then click Remove.
Remove all breakpoints simultaneously.	a. Click the **Breakpoints** tab to show the Breakpoints view. b. Click the **Remove All Breakpoints** icon on the toolbar, or right-click any breakpoint, then click Remove All.

What to do next

If you have finished debugging, continue with: “Debug: ending a session” on page 683.

Stepping through message flow instances in the debugger

After you have added one or more breakpoints to a message flow in the debugger, you can step through the flow, pausing as required.

Before you begin

Add one or more breakpoints to your message flow. See: “Adding breakpoints in the flow debugger” on page 669.

About this task

The message flow debugger pauses flow processing at the first breakpoint it encounters. You can then continue with one or more of the following tasks, as appropriate:

- "Debugging: resuming message flow processing" on page 673
- "Debugging: running to completion" on page 674
- "Debugging: stepping over nodes" on page 674
- "Debugging: stepping into subflows" on page 675
- "Debugging: stepping out of subflows" on page 675
- "Debugging: stepping through source code" on page 676

What to do next

As you step through the message flow you can look at the processing data. When you have finished, end your debugging session:

- "Debugging data" on page 677
- "Debug: ending a session" on page 683

Debugging: resuming message flow processing
Each time message flow processing pauses at an active breakpoint, you can investigate the state of the flow, then resume processing.

Before you begin

Add one or more breakpoints to your message flow; see "Adding breakpoints in the flow debugger" on page 669.

About this task

When message flow processing is paused at a breakpoint, you can resume processing:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Resume Flow Execution**  on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Resume**  .

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If no further enabled breakpoint exists at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If message flow debugging is complete, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Debugging: running to completion
When message flow processing pauses at an active breakpoint, you can let it continue to the end of the flow, ignoring other breakpoints.

Before you begin

Add one or more breakpoints to your message flow; see "Adding breakpoints in the flow debugger" on page 669.

About this task

When message flow processing is paused at a breakpoint, you can restart processing so that the message flow runs to completion.

If you want the flow to continue processing, but you want to pause at the next enabled breakpoint instead of running to completion, see "Debugging: resuming message flow processing" on page 673.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Run to completion** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Run to completion**.

Results

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is removed automatically from the Debug view.

What to do next

If debugging is complete, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Debugging: stepping over nodes
When message flow processing pauses at an active breakpoint, you can step over the node and continue processing until the next active breakpoint, ignoring breakpoints that are set in code in the node.

Before you begin

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 669.

About this task

To step over the next node and continue message flow processing:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Step Over Node** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Over Node**.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If no further enabled breakpoint exists at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If debugging is complete, you can remove the breakpoints, or end the debugging session:

- “Removing breakpoints in the flow debugger” on page 671
- “Debug: ending a session” on page 683

Debugging: stepping into subflows

When message flow processing pauses at a breakpoint, you can step into the subflow that follows.

Before you begin

Add one or more breakpoints to your message flow; see “Adding breakpoints in the flow debugger” on page 669.

About this task

To step into a subflow, complete the following steps.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Step Into Subflow** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Into Subflow**.

Results

The subflow opens in the Message Flow editor and displaces the parent message flow. Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If no further enabled breakpoint exists at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If debugging is complete, you can remove the breakpoints, or end the debugging session:

- “Removing breakpoints in the flow debugger” on page 671
- “Debug: ending a session” on page 683

Debugging: stepping out of subflows

When message flow processing is paused at a breakpoint in a subflow, you can step out of the subflow.

Before you begin

Complete the following tasks:

- “Adding breakpoints in the flow debugger” on page 669
- “Debugging: stepping into subflows” on page 675

About this task

To step out of a subflow, complete the following steps.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Step Out of Subflow** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Out of Subflow**.

Results

The debugger continues processing until it reaches the connection from the output terminal of the subflow, where it pauses. The parent flow opens in the Message Flow editor, displacing the subflow.

What to do next

If debugging is complete, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Debugging: stepping through source code

When message flow processing is paused at a breakpoint on entry to a node that contains ESQL code or Java code, you can step through the code.

Before you begin

Add one or more breakpoints to your message flow; see "Adding breakpoints in the flow debugger" on page 669.

About this task

The nodes that can contain ESQL code or Java code are listed in "Flow debugger overview" on page 660.

To step through your source code, complete the following steps.

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, click **Step into Source Code** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Into**.
3. When message flow processing is paused at a breakpoint in the ESQL or Java code, you can step through the source code line by line. In the Debug view, click **Step Over** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Over**.

   Repeat this step as often as necessary.

   A single line of source code runs and the flow pauses at the next line of code. What you can do depends on what type of code is contained within the node. For more information, see on of the following topics:

   - "Debugging ESQL" on page 678
   - "Debugging Java" on page 679

   If the debugger is paused before the last line of code when you step over, the last line of code runs and message flow processing continues until the next breakpoint in the logical processing of the current message. If no further enabled breakpoint exists at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

4. If you finish looking at the code before the last breakpoint, you can continue processing the message flow. In the Debug view, click **Step Return** on the toolbar.

   Alternatively, right-click the flow stack frame, then click **Step Return**.

   The source code runs to completion from the current breakpoint and message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If no further enabled breakpoint exists at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If message flow debugging is complete, you can remove the breakpoints or end the debugging session:

- “Removing breakpoints in the flow debugger” on page 671
- “Debug: ending a session” on page 683

Debugging data

You can view (and change) data in messages, ESQL code, or Java code as debugging progresses.

About this task

When you have added one or more breakpoints to a deployed message flow, the debugger stops the message flow processing at each breakpoint. Depending on the context of the breakpoint, you can do one of the following tasks:

- “Debugging messages” on page 677
- “Debugging ESQL” on page 678
- “Debugging Java” on page 679

Results

When you have finished debugging a message flow, you can remove the breakpoints, or end the debugging session:

- “Removing breakpoints in the flow debugger” on page 671
- “Debug: ending a session” on page 683

Debugging messages

When message flow processing has paused at a breakpoint in your message flow, you can examine and modify the message content.

Before you begin

Add one or more breakpoints to your message flow. See: “Adding breakpoints in the flow debugger” on page 669.

About this task

To examine and modify message data:

Procedure

1. Switch to the Debug perspective.
2. View the messages in the Variables view.

   The Breakpoints view and the Variables view share the same pane. Click the tab at the bottom to select the view that you want.

3. To alter a message, right-click it and select an option from the menu.

   You cannot alter the content of exceptions within a message.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Debugging ESQL

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains ESQL code, you can examine and modify the ESQL variables in the Flow Debugger.

Before you begin

Complete the following tasks:

- "Adding breakpoints in the flow debugger" on page 669
- "Debugging: stepping through source code" on page 676

About this task

You can browse ESQL variables in the Variables view in the Debug Perspective, and change their associated data values. You can also set breakpoints on lines in the ESQL code. See the following sections for further details:

- "Using breakpoints on ESQL code lines" on page 678
- "Working with ESQL variables" on page 678

Using breakpoints on ESQL code lines

Procedure

1. Switch to the Debug perspective.
2. Open the ESQL editor.
3. Right-click a line where you want to set a breakpoint.

   You cannot set a breakpoint on a comment line or a blank line.
4. Select from the menu to create, delete, or restrict the breakpoint, in a similar way to normal debugger breakpoints, as described in "Working with breakpoints in the flow debugger" on page 669.

Working with ESQL variables

Procedure

1. Switch to the Debug perspective.
2. Open the Variables view.

   Variables are shown in a tree, using the symbol  .

3. To work with a variable, right-click it and select an option from the pop-up menu.

   You cannot update message trees, or REFERENCE variables.

   For example, if you have declared the following ESQL variables, you can change their values in the debugger:

```
DECLARE myInt INT 0;
DECLARE myFloat FLOAT 0.0e-1;
DECLARE myDecimal DECIMAL 0.1;
DECLARE myInterval INTERVAL DAY TO MONTH;
```

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Debugging Java

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains Java code, you can examine and modify the Java variables in the Flow Debugger.

Procedure

1. To open the Command Console, click **Start** > **All Programs** > **IBM App Connect Enterprise 12.0.n** > **IBM App Connect Enterprise Console 12.0.n**.
2. Start the integration node by running the **mqsistart** command in the Command Console.
3. Check that the Java debug port is set by running the **mqsireportproperties** command (all on one line) in the Command Console:

   ```
   mqsireportproperties integrationNodeName -e integrationServerName -o ComIbmJVMManager -r
   ```

   For example:

   ```
   mqsireportproperties TEST -e default -o ComIbmJVMManager -r
   ```

4. Set the Java debug port by running the **mqsichangeproperties** command (all on one line) in the Command Console:

   ```
   mqsichangeproperties integrationNodeName -e integrationServerName
   -o ComIbmJVMManager -n jvmDebugPort -v port_number
   ```

   For example:

   ```
   mqsichangeproperties TEST -e default
   -o ComIbmJVMManager -n jvmDebugPort -v 3920
   ```

5. Stop and restart the integration node by running the **mqsistop** and **mqsistart** commands.
6. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Application Development view.
7. Add a breakpoint where the Java method is called, by following the instructions in "Adding breakpoints in the flow debugger" on page 669.
8. To step directly into the Java code during the debugging process, add a breakpoint in the Java code.

9. Deploy the BAR file that includes the JAR file that contains the Java code, by following the instructions in "Deploying integration solutions to a production environment" on page 2480.
10. Click **Run** > **Debug** to open the **Debug** wizard.
11. Right-click **Debug** in the list of elements on the left and click **New**.
12. Set the **Java Debug Port** with the same value that you specified for the **-v** parameter on the **mqsichangeproperties** command, and click **Apply** to save your changes.
13. Click the **Source** tab, specify the source file location, and click **Apply** to save your changes.
14. Click **Debug** to start the debug process.

Working with Java variables

About this task

When message flow processing has paused at a breakpoint in the source code within a node that contains Java code (a user-defined node or a JavaCompute node), you can browse Java variables in the Variables view on the Debug perspective, and change their associated data values.

Procedure

1. Switch to the Debug perspective.
2. Click the **Variables** tab to open the Variables view if it is not already open.

   Variables are shown in a tree, using the symbol  .
3. To work with a variable, right-click it and select an option from the menu.

Results

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

What to do next

When you have completed debugging the message flow, you can remove the breakpoints or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 671
- "Debug: ending a session" on page 683

Managing flows and flow instances during debugging

During a debugging session, there are various administrative tasks that you might need to do, which include detaching the debugger from the integration server when you have finished.

About this task

When you have started a session for message flow debugging, you might want to complete one or more of the following associated tasks:

- "Debug: querying an integration server to find deployed flows" on page 681
- "Debug: stopping a message flow instance" on page 681
- "Debug: redeploying a message flow" on page 682
- "Debug: ending a session" on page 683

Debug: querying an integration server to find deployed flows
You can find the message flow that you want to work with in the Flow Debugger by refreshing the list of available flows.

About this task

During an active debugging session, you can query an integration server to find out what flows are currently deployed to it. The displayed list of message flows that are available in that integration server is updated. The updated list might include message flows that were not previously deployed, or that were not accessible because the flow was already being accessed by another developer.

To query an integration server for deployed flows:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, select the integration server that you want to query, then take one of the following actions:
   - Click **Refresh Selected Flow Engine to Get More Flow Types**  on the toolbar.
   - Right-click the integration server, then click **Refresh**  .

Results

The Debug view is refreshed with the names of the flows that are currently deployed to the integration server and are available.

What to do next

You can continue your debugging session and debug one of the listed message flows, or end your debugging session:

- “Working with breakpoints in the flow debugger” on page 669
- “Debug: ending a session” on page 683

Debug: stopping a message flow instance
While debugging, a message flow cannot be redeployed until it has been stopped.

About this task

While you are debugging, you might need to stop a message flow instance. For example, you might want to correct an error in your flow or source code. To do this, you must stop the flow and then redeploy it. See “Debug: redeploying a message flow” on page 682.

To stop message flow processing, run it to completion:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Run to completion**  on the toolbar.
   - or, right-click the flow stack frame, then click **Run to completion**  .

Results

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is automatically removed from the Debug view.

What to do next

After stopping a flow instance, you can start to debug another message flow, or end your debugging session:

- "Attaching the flow debugger to an integration server for debugging" on page 665
- "Debug: ending a session" on page 683

Debug: redeploying a message flow

If you want to change your message flow while you are debugging it, you must redeploy it to the integration server, then reattach the flow debugger.

Before you begin

Stop the message flow before you redeploy it. See: "Debug: stopping a message flow instance" on page 681

About this task

During your debugging session, you might find a problem in a message flow that you want to correct or see a behavior that you want to change. You can alter the flow to resolve the situation and redeploy the flow to the integration node:

Procedure

1. Switch to the Debug perspective.
2. Detach the debugger from the integration server by clicking **Detach from the Selected Flow Engines**  on the toolbar.
3. Switch to the Integration Development perspective.
4. Edit the flow in the Message Flow editor and save your changes.
5. Double-click the BAR file that contains your flow. Remove the flow, then add your edited version and save your changes.

   See "Adding resources to a BAR file" on page 2470.
6. Deploy your BAR file.

   Drag your BAR file from the Application Development view to the integration server in the Integration Explorer view. Check the Administration log to make sure that the deployment was successful.

   See: "Deploying integration solutions to a production environment" on page 2480.
7. Switch to the Debug perspective.
8. Reattach the debugger to the integration server.

   Click the down-arrow on the **Debug** icon  on the toolbar, and select **Debug** to invoke the **Debug (Create, manage, and run configurations)** wizard, and attach the flow engine again, following the instructions in "Attaching the flow debugger to an integration server for debugging" on page 665.

Results

The modified message flow is now deployed to the integration node, and the debugging session is ready for you to debug the new flow logic.

What to do next

Continue to use these tasks to debug your message flow:

- "Working with breakpoints in the flow debugger" on page 669
- "Stepping through message flow instances in the debugger" on page 672

Debug: ending a session
Finish debugging by detaching the flow debugger from the integration server to which your message flows are deployed.

About this task

When you have finished debugging a flow, detach the flow debugger from the integration server. Other developers are then able to attach the debugger to the integration server. Detaching the flow debugger also restores the performance of your workbench environment, which might have been reduced by having the debugger attached.

To detach the flow debugger from an integration server:

Procedure

1. Switch to the Debug perspective.
2. In the Debug view, select the name of the integration server from which you want to detach the flow debugger, then take one of the following steps:
   - On the toolbar, click **Detach from the Selected Flow Engines**  .
   - Right-click the integration server, then click  **Detach**).

Results

All existing flow instances are automatically run to completion and the flow debugger is detached from the integration server. Your debugging session is now finished. You can start a new debugging session at any time.

Debugging message flows that contain WebSphere Adapters nodes

You can use various methods to monitor message flows that include WebSphere Adapters nodes.

About this task

Before you use any of the methods listed in this section, ensure that the appropriate JAR files and shared libraries are available to the WebSphere Adapters nodes. For more information, see "Preparing the environment for WebSphere Adapters nodes" on page 209.

Also, check for the latest information about WebSphere Adapters; see WebSphere Adapters technotes.

- **User and service trace:** You can use user and service trace to trace a message flow that contains WebSphere Adapters nodes. For more information, see "Using trace" on page 3026.
- **Flow debugger:** Use the flow debugger in the normal way to debug a message flow that contains WebSphere Adapters nodes. For more information, see "Testing your message flow by using the flow debugger" on page 659.
- **Adapter event table:** The WebSphere Adapters nodes use an event table to communicate the outcome of operations asynchronously to a calling application. For more information, see "Creating a custom event project in PeopleTools" on page 1127.

Handling exceptions that are raised by a WebSphere Adapters request node

The WebSphere Adapters request nodes raise exceptions that indicate the following Enterprise Information System (EIS) failures.

Message number	Exception type	Explanation
BIP3511	RecordNotFound	The requested record could not be found in the EIS.
BIP3512	DuplicateRecord	An attempt was made to create a record that already exists in the EIS.
BIP3513	MultipleMatchingRecords	A retrieve request matched more than one record. To retrieve multiple records, perform a `retrieveall` operation.
BIP3515	MatchesExceededLimit	A `retrieveall` exception returned more entries than the maximum allowed number.
BIP3516	MissingData	The message tree that was sent to the adapter request node does not have all the required fields set.

If an exception occurs that does not fit into the categories in the table, the node raises a general BIP3450 message that describes the problem.

You can use these exceptions to perform special processing when you do not want the exceptions to be treated as errors. For example:

- If a `create` operation fails because the record already exists, you could modify the request to an `update`.
- If a `retrieve` operation fails because the request matches more than one record, you could try a `retrieveall` operation instead.
- If a `retrieve` operation fails because the record could not be found, an empty record could be returned.

To handle these exceptions, you can connect a message routing node, Compute node, or JavaCompute node to the Failure terminal of the WebSphere Adapters request node, and route the exception to other processing nodes based on the exception message number.

XSD Schema Validation problem

When you configure an SAP adapter in the IBM App Connect Enterprise Toolkit, you might see the following warning, referring to an unresolvable IBM XML schema:

```
CTDX1101W : XSD: The location '' has not been resolved
example1.xsd /EAI_ESB_LIB line 2
example2.xsd /EAI_ESB_LIB line 2
XSD Schema Validation Problem
```

This warning is caused by the following namespace reference in an `<xsd:import>` element:

```
<xsd:import namespace="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/metadata"/>
```

No types or elements from this namespace are referenced in the logical structure of the XSD. An XML schema that references this namespace is in the referenced connector project, and is pointed to from the XML catalog. The XML catalog entry is populated when the SAP connector project is brought into the workspace.

This warning is benign and can be safely ignored, because it does not affect the structural content of your schema definition.

Tuning the SAP adapter for scalability and performance

You can monitor the performance of a message flow that contains SAP nodes by using user trace, and accounting and statistics data, then use that information to tune your flow by configuring an appropriate number of listeners and additional instances for the message flow.

Testing your message flow by adding Trace nodes

By adding a Trace node to a message flow, you can write debugging messages to a file, to user trace, or to the system log, and review those messages after the message flow has processed some data.

About this task

You must add a Trace node when the message flow is designed. "Viewing the logical message tree in trace output" on page 685 explains how to view the structure of the logical message tree at any point in the message flow, and contains an example of the message content. You can turn the Trace node off when a message flow is promoted to production to improve performance, but you can turn the node on when required. Performance can be affected when Trace nodes are active. The extent to which performance is affected depends on the destination that you choose for the debugging messages; for example, writing to user trace is typically faster than writing to a file or to the system log.

The debug messages can include writing part or all of the logical message tree, but they can also include hard-coded strings to identify a particular point in the message flow (such as PRINTF in a C program). If you write the entire message tree in a Trace node, the behavior of the message flow might be changed. Typically, only the parts of the message that are referenced are parsed, rather than the entire message.

Procedure

- Add a Trace node to your message flow, then set the following properties on the node (as described in detail in node):
  - Set the destination of the trace record that is written by the node to `User Trace`, `Local Error Log`, or `File`.
  - If you choose `File`, set the file path of the file to which to write records.
  - Use the `Pattern` property to create an ESQL pattern that specifies the data to be included in the trace record.
  - Specify the message catalog from which the error text for the error number of the exception is extracted.
  - Specify the error number of the message that is written.
  -
- After you have added a Trace node to your message flow, you can turn it on or off, as described in "Switching Trace nodes on and off" on page 3038.
- To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all of the message (including headers and all four message trees) to the trace output destination. The following topics describes how to view that output: "Viewing the logical message tree in trace output" on page 685

Viewing the logical message tree in trace output

To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all the message (including headers and all four message trees) to the trace output destination.

About this task

You might find trace output useful to check or record the content of a message before and after a node has changed it, or on its receipt by the input node. For example, if you include a Compute node that builds a destination list in the local environment tree, you might want a record of the structure that it has created as part of an audit trail, or you might want to check that the Compute node is working as you expect it to.

UNIX On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the newline character. To record a large amount of data in a log on UNIX, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow for which you want to view messages.

   Open an existing message flow, or create a message flow.
3. Include a Trace node wherever you want to view part or all the message tree structure.

   You can include as many Trace nodes as you choose; however, each node that you introduce can affect the performance of message flow processing.
4. Set the Trace node properties to trace the message, or parts of the message, that you want to view.

   Specify the parts of the message by using ESQL field references. Several examples are included later in this topic.
5. If you have added a Trace node to investigate a particular behavior of your message flow, and have now resolved your concerns or checked that the message flow is working correctly, remove the Trace node or nodes, and redeploy the message flow.

Example

Assume that you have configured a message flow that receives an XML message on an IBM MQ queue in an MQInput node. The input message includes an MQRFH2 header. The message has the following content:

```
<Trade type='buy'
 Company='IBM'
 Price='200.20'
 Date='2000-01-01'
 Quantity='1000'/>
```

You can include and configure a Trace node to produce output that shows one or more of the trees created from this message: the message body, environment, local environment, and exception trees. If you choose to record the content of the message body, the Properties tree and the contents of all headers (in this example, at least an MQMD and an MQRFH2) are included. You specify what you want to be recorded when you set the Trace node property `Pattern`. You can use most of the correlation names to define this pattern (you cannot use those names that are specific to the Compute node).

Message body

If you want the Trace node to write the message body tree including Properties and all headers, set `Pattern` to ${Root}. If you want only the message data, set `Pattern` to ${Body}.

The trace output generated for the message tree of the preceding message with `Pattern` set to ${Root} would look like the following example:

```
Root
      Properties
         CreationTime=GMTTIMESTAMP '1999-11-24 13:10:00'          (a GMT timestamp field)

  ... and other fields ...

      MQMD
         PutDate=DATE '19991124'                                  (a date field)

         PutTime=GMTTIME '131000'                                 (a GMTTIME field)

  ... and other fields ...

      MQRFH
          mcd
          msd='xml'                                              (a character string field)

         .. and other fields ...

      XML
         Trade
          type='buy'                                             (a character string field)

         Company='IBM'                                           (a character string field)

         Price='200'                                             (a character string field)

         Date='2000-01-01'                                       (a character string field)

         Quantity='1000'                                         (a character string field)
```

Environment

To trace any data in the environment tree, set `Pattern` to ${Environment}. This setting produces output like the following example:

```
 (0x1000000)Environment = (
      (0x1000000)Variables = (
        (0x1000000)MyVariable1 = (
          (0x2000000) = '3'
        )
        (0x1000000)MyVariable2 = (
          (0x2000000) = 'Hello'
        )
      )
  )
```

To trace particular variables in the variables folder of the environment tree, you can use a more specific pattern, for example ${Environment.Variables.MyVariable1}. This setting returns the value only (for example, it returns just the value 3).

LocalEnvironment

To trace data in the local environment tree, set `Pattern` to ${LocalEnvironment}. The output you get is like the following example, which shows that a destination list has been created in the local environment tree:

```
(0x1000000)Destination = (
    (0x1000000)MQ                = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
     (0x1000000)MQDestinationList = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'OLDMQOUT'
      )
    )
    (0x1000000)RouterList        = (
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'continue'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'custdetails'
      )
      (0x1000000)DestinationData = (
        (0x3000000)labelname = 'trade'
      )
    )
  )
```

Another example, shown here, includes a WrittenDestination folder. This example represents a trace that has been written by a Trace node that is included after an MQOutput node, where the Out terminal of the MQOutput node is connected to a sequence of nodes including the Trace node. When an Out terminal is connected, the local environment is augmented with information about the action that the output node has performed.

```
(0x1000000)Destination        = (
    (0x1000000)MQ                = (
      (0x1000000)DestinationData = (
        (0x3000000)queuename = 'MQOUT'
      )
    )
  (0x1000000)WrittenDestination = (
    (0x1000000)MQ = (
      (0x1000000)DestinationData = (
        (0x3000000)queueName        = 'MQOUT'
        (0x3000000)queueManagerName = 'MQSI_SAMPLE_QM'
        (0x3000000)replyIdentfier   = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)msgId            = X'414d51204d5153495f53414d504c455f1f442f3b12600100'
        (0x3000000)correlId         = X'0000000000000000000000000000000000000000000000000000'
        (0x03000000):GroupId          = X'414d512042524f4b45523220202020203f59934620001803'
        (0x03000000:NameValue):bindingType             = 'CLIENT' (CHARACTER)
        (0x03000000:NameValue):destinationQueueManager = 'QM1_REG1' (CHARACTER)
        (0x03000000:NameValue):queueManagerHostname    = 'MYQUEUEMANAGERHOSTNAME' (CHARACTER)
        (0x03000000:NameValue):listenerPortNumber      = 9000 (INTEGER)
        (0x03000000:NameValue):channelName             = 'MYCHANNELNAME' (CHARACTER)
      )
    )
  )
)
```

ExceptionList
To trace data in the exception list, set `Pattern` to ${ExceptionList}.

You can also view message structure in the message flow, and other information, when you use the flow debugger.

Enabling the exception log

The exception log enables the diagnosis of exceptions that occur at any time during processing on an integration server.

About this task

The exception log resource manager enables data relating to exceptions that occur at any time during processing on an integration server to be recorded, in an exception log, so that the message flow developer or administrator can diagnose such exceptions. The exception log is particularly useful when you are trying to diagnose exceptions that are not reported to the error log because the exception has

been handled in a message flow. The exception log is controlled by three properties that are available in the `server.conf.yaml` configuration file. The values of these properties can be viewed by using the **mqsireportproperties** command.

Even if the exception is not present in the error log meaning that it has been handled by flow logic, the exception is added to the exception log.

The exception log is written to one of the following locations depending on whether the integration server is independent or is managed by an integration node:

- `WorkDirectory/config/common/log/integration_server.`*`IntegrationServerName`*`.exceptionLog.txt` when the integration server is independent.
- `MQSI-REGISTRY/common/log/`*`IntegrationNodeName`*`.`*`IntegrationServerName`*`.exceptionLog.txt` when the integration server is managed by an integration node.

The exception log is a rolling collection of 4 files; when a file is full, data is added to the next file. When the last file has been filled, data starts to be added to the first file, overwriting the previous data there. This cycle of writing continues, ensuring that the newest data is retained. Restarting the integration server causes the previous log to be moved and a numeric suffix added to it.

The exception log includes:

- A summary or short description of the exception.
- A list of the exception's inserts.
- Message numbers of any nested exceptions.
- Optionally, details recorded by the Flow Thread Reporter. If the exception occurs in a message flow, the Flow Thread Reporter provides data to enable you to identify the location in the message flow at which the exception occurred.

Exceptions are listed in the order of their creation.

Procedure

Enable the exception log by amending properties in the `ExceptionLog` subsection of the `ResourceManagers` section of the `server.conf.yaml` configuration file.

The parameters are:

enabled
Controls whether the exception log is enabled.

Valid values are `true` and `false`.

The default value is `false`.

-n includeFlowThreadReporter
Controls whether, if an exception occurs on a message flow thread, additional information from the Flow Thread Reporter is added to the exception log. The additional information includes the flow thread history and stack. This information is quite verbose but is useful if you want to identify the precise location in a flow that an exception occurred, or what the passage through the flow was.

Valid values are `true` and `false`.

The default value is `false`.

showNestedExceptionDetails
Controls whether the exception log includes the details of any nested exceptions.

Valid values are `true` and `false`.

By default, the exception log does not include the details of any nested exceptions because they will already have been printed earlier in the log. Setting this option to `true` results in nested exceptions being re-printed, which is useful if your exception log has multiple concurrent exceptions. However, it might not be obvious which previous entry relates to the current exception.

To enable the exception log and set it to include details of any nested exceptions and additional information from the Flow Thread Reporter, specify the parameters in the `server.conf.yaml` file as follows:

```
ExceptionLog:
    enabled: true                    # Enables logging of exceptions to <workdir>/config/
common/log/integration_server.<name>.exceptionLog.txt
    includeFlowThreadReporter: true   # Toggles whether exception in the exception log include a
 flow stack and history from the flow thread reporter
    showNestedExceptionDetails: true  # Toggles whether nested exceptions are shown by default
 in the exception log
```

Message flow behavior

Message flow behavior is initially defined by IBM App Connect Enterprise, but there are some aspects of behavior that you can modify.

When a message is received and processed by a message flow, App Connect Enterprise handles both successful and failed processing. The default actions, which are independent of the protocol that you are using to exchange messages with your message flows, are described in "Default message flow behavior" on page 690.

If the default behavior does not meet your processing requirements, you can change some aspects of the behavior by tuning or reconfiguring your message flows.

- Learn about the default behavior supported by App Connect Enterprise.
- Review the options that are available to you if you want to change the behavior.

Default message flow behavior

IBM App Connect Enterprise controls the behavior of your message flows, and defines the default actions that are taken if you do not modify the message flow behavior.

The following default actions are provided for all messages that are processed by your message flows:

The execution and threading model
When you deploy a message flow to an integration server, App Connect Enterprise allocates a certain number of threads to that message flow. For further information about how this processing works, see "Execution and threading models in a message flow" on page 691.

Error handling
When you configure your message flows to process messages, you can include error processing of your own. App Connect Enterprise runs default processing if an error occurs for which you have not provided additional processing; this processing is independent of the protocol that you are using for communication. It is described in "Default error handling" on page 691.

Transactional support
App Connect Enterprise handles all messages individually, and does not relate an action taken by the message flow to other actions, processes, or messages. A message flow that is not configured to implement transactional support succeeds or fails regardless of the outcome of interactions with, or updates to, other resources such as IBM MQ queues and databases. Learn more about transactional support in "Message flow transactions" on page 691.

Data conversion
Different hardware platforms and operating systems operate in different code pages. Operating system code pages are also affected by the locale in which you operate. App Connect Enterprise does not allow for these differences unless you change some aspects of its configuration. If you are exchanging messages between unlike systems, you might have to update App Connect Enterprise or IBM MQ configuration, or design and supply your own conversion procedures.

For information about how you can change the configuration of App Connect Enterprise, its IBM MQ queue manager, and your message flows to influence these behaviors, see "Changing message flow behavior" on page 696.

Execution and threading models in a message flow

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

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

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

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

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

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

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

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

Default error handling

IBM App Connect Enterprise provides basic error handling for all your message flows.

When an exception is detected within a message flow node, the message and the exception information are propagated to the Failure terminal on the node. If the node does not have a Failure terminal, or is not connected, App Connect Enterprise throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node.

The actions taken by the input node are protocol-dependent; if your message flow starts with an MQInput, its error handling is different from the error handling provided by a FileInput node. For more information, review the terminals and properties in the reference topic for the input node type.

The actions also depend on whether you have configured the message flow to be transactional. Some message flow nodes support transactions; others are non-transactional. If the flow is transactional, the message is returned to its source; for example, it is restored on the IBM MQ queue. For an overview of IBM MQ connection behavior, see "IBM MQ topologies" on page 739.

If basic error processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling. For example, you can add in a sequence of nodes to deal with one or more errors that you might expect to occur in your flow. You can also configure your flow to handle unexpected errors (exceptions). For more details about the actions that you can take, see "Handling errors in message flows" on page 1883.

Message flow transactions

A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as *consistency*: transactions might have other properties of *atomicity*, *isolation*, and *durability*.

IBM App Connect Enterprise supports transactions, and every piece of data processed by a message flow has an associated transaction. A message flow transaction is started when input data is received at an

input node in the flow; it is committed when the flow has finished with that message, or rolled back if an error occurs.

If the flow contains more than one input node, one transaction is started for each input node when it receives input data. A transaction is started for every type of input node, including user-defined input nodes.

The message flow nodes that you include in your message flow provide specific processing of a message, according to the defined function of each node. The processing that they do includes internal work, some of which you can influence by configuring the node properties. Some message flow nodes perform additional tasks that might affect systems that are external to the message flow, integration server, or integration node.

If an external system, such as a database, supports the concept of commit and rollback, and it can take part in an App Connect Enterprise transaction, you can configure the message flow node so that the work it does is included in the flow transaction. Depending on the message flow node, you can also specify if the work done in an external system that supports transactions is committed immediately, or when the message flow transaction completes.

Many of the resources with which your message flows can interact are controlled by resource managers that can participate in coordinated transactions; for example, databases, IBM MQ messages and queues, CICS, and JMS messages. Other resource managers do not provide transactional support; for example, the HTTP protocol and file systems.

Commit or rollback

If the resource can participate in a transaction, you can configure it so that the work it does is committed or rolled back only when the message flow completes, or when the node completes. Databases and IBM MQ queues are examples of resources that you can use in this way. If the resource does not have transactional behavior, all the work that it does is committed immediately. For example, files and HTTP connections do not support transactions.

Updates that are made by a message flow are committed when the flow processes the input message successfully. The updates are rolled back if the following conditions are met:

- A node in the flow throws an exception that is not caught by a node other than the input node (for example, the node itself, or a TryCatch node)
- The Catch terminal of the input node is not connected
- The Catch terminal of the input node is connected, but an unhandled exception occurs in the message flow nodes that are connected to the Catch terminal.

On distributed systems, message flow transactions are managed by App Connect Enterprise, by default. These transactions are known as *local transactions* or *locally coordinated transactions*. When control returns to the input node when the flow finishes processing, the input node either commits or rolls back the operations that have been taken, excluding the individual nodes that have been configured to perform their own commits and rollbacks, or that have no support for this option.

If more than one resource is accessed by the message flow, an error might occur that prevents all the resources committing all the work that has been done. App Connect Enterprise raises an exception and handles exception processing in a way that is determined by the transport that is involved. For example, messages that were read from IBM MQ queues are restored to those queues, and fault messages are sent to applications that submitted a message across HTTP (because HTTP has no concept of rollback). As a result of these actions, the status of the resources might become inconsistent.

If it is important that your data and operations remain consistent, and that all operations are committed, or rolled back if one or more operations fail, you can coordinate the activity of the message flow.

Coordinating transactions

Coordination is provided by an external transaction manager, which uses XA protocols to interact with resource managers. The transaction manager is called by the input node when the message flow has

concluded (successfully or with errors). The transaction manager, rather than the input node and the integration node, interacts with the relevant resource managers to initiate the correct actions for each resource. Transactions that are controlled by a transaction manager in this way are known as *globally coordinated transactions*.

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that App Connect Enterprise requires access to IBM MQ when processing messages. The queue manager must be local, and configured to be the transaction manager. All other queue managers that interact with MQ nodes in that message flow are treated as locally-coordinated resources. For more information about using IBM MQ with App Connect Enterprise, see “Enhanced flexibility in interactions with IBM MQ ” on page 26.

The role of the transaction manager

The result of the actions taken by message flows is the same for both local and globally coordinated transactions if the message flow is successful in all its actions. The advantage of a globally coordinated transaction is the ability to ensure that either all actions are committed, or none.

The external transaction manager, which operates a two-phase commit strategy, supports cases where one or more external resource managers are temporarily unavailable during commit processing. This potentially small window for failure might be costly for your business environment; the external transaction manager helps to eliminate the occurrence of a failure window. Therefore, the decision to include an external transaction manager, which involves a performance overhead, is an administrative decision, not one to be taken at message flow design time.

An external transaction manager does not prevent message loss; even if you use transaction coordination, you must configure and code your message flows to handle potential errors as much as you can.

To configure a message flow to be globally coordinated, you must also set up your environment so that your resource managers are defined to the supported transaction manager. On distributed systems, transactions can be coordinated by IBM MQ.

This configuration might require you to change settings in the transaction manager as well as the participating resource managers.

Database access modes and locks

You must use separate ODBC connections if you want to include message flow nodes with Automatic transaction status and nodes with Commit transaction status in the same message flow, where the nodes operate on the same external database. Set up one connection for the nodes that are not to commit until the completion of the message flow, and a second connection for the nodes that are to commit immediately.

- If nodes with Commit transaction status are followed by a node with Automatic transaction status, the nodes with Commit transaction status commit independently of the flow transaction, and the nodes with Automatic transaction status commit at the end of the flow.
- However, if nodes with Automatic transaction status are followed by a node with Commit transaction status, and you do not use separate ODBC connections, error message BIP4001 is issued, because otherwise the node with Commit transaction status commits the work of the Automatic nodes prematurely.

Linux Windows UNIX On distributed systems, individual relational databases might not support this mode of operation.

If you define more than one ODBC connection to the same data source, you might get database locking problems. In particular, if a message flow node with Automatic transaction status carries out an operation, such as an INSERT or an UPDATE, that causes a database object (such as a table) to be locked, and a subsequent node tries to access that database object by using a different ODBC connection, an infinite lock (deadlock) occurs.

The second message flow node waits for the lock acquired by the first to be released, but the first node does not commit its operations and release its lock until the message flow completes. The flow cannot complete because the second node is waiting for the database lock held by the first node to be released.

Such a situation cannot be detected by a DBMS automatic deadlock-avoidance routine because the two operations are interfering with each other indirectly by using the integration node.

You can use either of two options to avoid this type of locking problem:

- Design your message flow so that uncommitted (automatic) operations do not lock database objects that subsequent operations access across a different ODBC connection.
- Configure the lock timeout parameter of your database so that an attempt to acquire a lock fails after a specified length of time. If a database operation fails because of a lock timeout, an exception is thrown that the integration node handles in the usual way.

For information concerning which database objects are locked by particular operations, and how to configure the lock timeout parameter of your database, consult your database product documentation.

The transactional model

The transactional model describes the way in which you can use transactions in message flows to accomplish certain tasks and results.

A message flow consists of the following constituent parts:

- An input source
- The message flow or logic, which is defined by a sequence of nodes
- Zero or more external resources that are accessed during the flow
- Zero or more output targets

The following steps represent a typical sequence of events in the message flow transaction:

1. A message is taken from the input source; for example, a queue.
2. Data is read from or written to one or more external resources; for example, a database.
3. A message is sent to an output target; for example, a queue.
4. The system quiesces and waits for the next input message.

During this sequence of events, the state of the data in the system changes, regardless of the number of external resources that the message flow accesses, and whether it generates an output message.

Consider the following diagram:

```
-----x---------x---x-------x-------------x----x-----
     1         2   3       4             5    6
```

The line represents the data in the system as time passes. At time 1, a message arrives and is taken from the input source. At times 2, 3, 4, and 5, data is used to update external resources; for example, a database table or an output queue. Changes in the state of the data are indicated in the diagram by the *x* symbol. At time 6, the output messages are sent and the system is inactive. Between these events, the state of the data does not change; this state is indicated in the diagram by the = symbol.

If a failure occurs on the system (for example, a loss of power to the computer on which the integration node is running), the changes to the state of external resources that were made before the failure have been implemented, but no more changes take place after the failure. This situation is unacceptable in certain circumstances; for example, if a system failure occurs when making a payment from a current account to a mortgage account, the payment might be taken from the current account, but it is not added to the mortgage account.

Transactions

To avoid the problem that is described previously, the integration node and the external resource managers with which it works, have a *transactional model*. The integration node starts a transaction when data is received by an input node in the message flow, and completes when the processing on that data

is finished. For more details about message flow transactions, see “Message flow transactions” on page 691.

As processing proceeds in a transaction, additional data is recorded that allows the original state to be restored in the event of a failure. The following diagram illustrates the state of this extra data:

```
-----x=========x===x=======x=============x====x-----
     1         2   3       4             5    6
```

The line in the diagram represents the extra data in the system as time passes. At time 1, input data arrives from the input source; for example, a queue. Before time 1, no extra data exists in the system; this state is indicated in the diagram by the - symbol. After time 1, the state represents the fact that data has been received from the input source, so that it can be restored, if necessary. At times 2, 3, 4, and 5, data is used to update external resources such as databases or files. Again, the state of the extra data changes so that the changes to those external resources can be undone, if necessary. At time 6, the output messages are sent, the system is inactive, and extra data in the system no longer exists.

Between these events, the state of the extra data does not change; this state is indicated by the = symbol. If a failure occurs between time 1 and time 6, the extra data is used to restore the original state of the data held by the external resources. Therefore, effectively, no output data has been written to the output target, none of the external resources have been updated, and the input data has not been received from the input source. If no failure occurs, the changes become permanent at time 6 (an undo operation that follows a subsequent failure will not undo the changes).

This mode of operation is known as *coordinated transaction mode*. The successful completion of a transaction is known as its *commit*. Unsuccessful completion is known as *rollback*.

Uncoordinated auxiliary transactions

The key feature of the coordinated transaction mode of operation is that, regardless of where or when the failure appears, either all of the changes to external resources that are associated with one input message are made, or none of the changes are made. However, this behavior is not always suitable, as the following examples illustrate:

- You want to create an audit log of all attempts at processing. The log entries must be committed, even when updates to other resources are rolled back.
- You want to send an acknowledgment or non-acknowledgment message back to the originator of the messages that you are processing, according to whether the message processing succeeds or fails. These messages must be sent even when the updates to other resources are rolled back.

If your message flows have requirements like these, you can configure message flows to change one or more resources in a separate, or auxiliary, transaction. Not all resource managers support this type of transaction.

For some resources, an auxiliary transaction is automatically started; for example, each database connection starts a transaction that is specific to that database, and all updates made in that transaction can be committed or rolled back.

The behavior of an auxiliary transaction is shown in the following diagram:

```
MAIN -----x=========x===x=======x=============x====x-----
          1         2   4       5             8    9

1st AUX --------------x======x========x------
                      3      6        7
```

The *MAIN* line represents the main transaction, which includes the extra data that is recorded to restore the original state if necessary. The *1st AUX* line represents an auxiliary transaction. At time 3, an external resource is updated, and another update is made at time 6. At time 7, the message flow determines that all the changes that must be made under the auxiliary transaction are complete, and it commits the changes.

If the message flow fails before time 7, the state of the system would be unchanged because both transactions would be rolled back. If failure occurs after time 7 but before time 9, the auxiliary transaction

would already have been committed. However, the main transaction would be rolled back. If a failure has not occurred by time 9, both transactions are committed.

Database auxiliary transactions

You can use more than one auxiliary transaction, and make a number of updates to database tables that can be committed or rolled back. You can then make additional changes to the same database tables, or to different tables, then commit or rollback these changes.

Each database that you use has its own auxiliary transaction; therefore, if the message flow updates tables that belong to different database instances (different data source names), an auxiliary transaction exists for each database. You can optionally commit or roll back these auxiliary transactions individually. Updates that have not been committed or rolled back when the message flow completes (at time 9 in the example shown previously) are committed or rolled back automatically by the integration node, according to whether the processing succeeded or failed.

Use the ESQL COMMIT and ROLLBACK statements to commit and roll back auxiliary database transactions. Obtain operations outside the main transaction by specifying the UNCOORDINATED keyword on the individual database statements (for example, the INSERT and UPDATE statements).

Queue auxiliary transactions

Not all queuing systems have the database capability that is described in the previous section. With IBM MQ, each individual uncoordinated read or write operation to a queue has an implied commit action. Therefore, you cannot put two messages, then decide to commit both or roll back both. The COMMIT and ROLLBACK statements therefore operate only on databases.

Nodes

The previous sections refer to message flows, but not to nodes. The way in which a message flow is divided into nodes has no effect on transactions. For operations on databases, an unlimited number of nodes can make updates to the main transaction, and to an unlimited number of auxiliary transactions, without restriction.

Changing message flow behavior

You can change the behavior that is taken by your message flows to process messages in different ways and at different times.

About this task

If one or more of the default behaviors that are described in “Default message flow behavior” on page 690 is not sufficient or appropriate for your message processing, you can change some characteristics of that behavior at different times during the design, development, and production cycles:

- “Behavior you can change when you design your message flows” on page 696
- “Behavior you can change when you deploy your message flows” on page 697
- “Behavior you can change at any time” on page 697
- “Behavior you can change during message flow processing” on page 698

Behavior you can change when you design your message flows

About this task

You can influence the behavior of your message flows when you set message flow node properties, and when you connect together the nodes that you select to run in that flow.

Transactional support

You can configure your message flows to handle your messages, and other data, in transactions. These options are described in “Configuring transactionality for message flows” on page 698.

Error handling
App Connect Enterprise provides an initial level of error processing for all message flows. If you require further support in your message flows, you can add your own processing. You can learn more about these options in “Handling errors in message flows” on page 1883.

The message flow nodes that support some protocols provide extra default error handling; this support is described, where relevant, in the sections in “Connecting client applications” on page 737.

Data conversion
If you are exchanging messages between unlike systems, you can update App Connect Enterprise or IBM MQ configuration, or design and supply your own conversion procedures. Options are described in “Configuring message flows for data conversion” on page 727.

User-defined properties
You can create user-defined properties for your message flows to associate values with those message flows. You can then configure the nodes in your message flows to access those properties and their values by coding ESQL or Java programs. Read more about user-defined properties, and see how to create them in the Message Flow editor.

Promoted properties
You can promote some message flow node properties to the level of the message flow in which the node is included. The advantages of this technique are described in “Promoted properties” on page 568; see “Defining a promoted property” on page 729 for details of how to use these properties.

Behavior you can change when you deploy your message flows

About this task

After you have decided on the message flow content, you can change some aspects of its operation before or after deployment.

Configurable properties and BAR overrides
Some of the properties on message flow nodes are configurable; that is, you can change their values when you deploy the message flow. By using this option, you can change some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager and data source information in the BAR file when you deploy it.

You can change these values by using the IBM App Connect Enterprise Toolkit or the **`mqsiapplybaroverride`** command.

The execution and threading model
You can increase the number of threads that your message flows use to reduce the time in which your messages are handled. You can also deploy multiple copies of a message flow to one or more integration nodes. For more information, see “Optimizing message flow throughput” on page 2763.

Behavior you can change at any time

About this task

The following technique can be implemented at any time in the solution lifecycle to enable developers, administrators, and operators to dynamically control certain aspects of message flow behavior.

Policies
A policy enables you to define a common approach to controlling certain aspects of message flow behavior, including flow rate, and particular node properties such as connection credentials. You can update a policy at any time in the solution lifecycle. For more information about policies, see “Overriding properties at run time with policies” on page 324.

Behavior you can change during message flow processing

About this task

If you want to change the behavior of your message flows in a more dynamic way, you can use the techniques that are described here. However, you must design your message flows so that they can take advantage of these additional services.

Policies

By using a policy, you can create and configure properties that relate to external services called by the integration node from your message flows.

Learn more about this option in "Overriding properties at run time with policies" on page 324.

Local environment overrides

You can configure some nodes to include your own processing; by coding ESQL, Java, or maps, you can modify the contents of the local environment tree within a message. Some fields in the local environment are used by nodes to determine how the message is processed, therefore by changing the tree contents, you can influence the behavior of subsequent nodes in the message flow. For more information about this option, see "Transforming and enriching messages" on page 1249.

Configuring transactionality for message flows

Process messages in either local or global transactions, by setting message flow node properties that determine how the resources that they represent participate within the transaction.

Before you begin

- Ensure that you understand how IBM App Connect Enterprise handles transactions, and the difference between local and globally coordinated transactions; see "Message flow transactions" on page 691.
- Create a message flow by following the instructions in "Creating a message flow" on page 574.

About this task

If you have configured an integration node that specifies a local MQ queue manager, you can set up globally coordinated transactions for the message flows that are managed by the integration node. The specified queue manager then performs the transaction manager role.

How individual message flow nodes, and the message flow itself, participate in transactions depends on the way you design and develop the message flow, and the level of additional configuration you perform.

1. You must configure the message flow node properties in your message flow to set the required level of participation in transactions.
2. If you want the updates that are made by the message flow to be globally coordinated by an external transaction manager (IBM MQ), you must configure the message flow properties.

   To use global transactions, IBM App Connect Enterprise must have access to IBM MQ. For more information about using IBM MQ with IBM App Connect Enterprise, see "Enhanced flexibility in interactions with IBM MQ " on page 26.

You can configure the message flow nodes in your message flow to determine how the work taken by each node participates in the message flow transaction. Most nodes for which transactionality is relevant have one or more properties that you can configure to dictate behavior. Therefore, you can decide for each individual message flow node whether it participates in the message flow transaction, or operates independently. Typically, these properties include an option of **Automatic**, so that subsequent nodes in the flow assume the characteristics set by the input node.

Message flow nodes that support transports that cannot participate in transactions might have other properties to determine what the integration node does when a message flow failure occurs. For example, the FileInput node has a set of **Retry** properties that you can set to determine failure behavior.

A few message flow nodes that interact with external resources do not provide properties; typically, these nodes are included in the message flow transactions. However, some exceptions exist. Review the

properties for each message flow node that you include in your flow to ensure that you understand what action is taken. For more information about each type of message flow node, see Built-in nodes.

If you configure a message flow node not to participate in the message flow transaction, the actions that it takes are committed, or rolled back, when the node exits. No further action is taken when the message flow itself completes.

To configure message flow behavior for transactions by setting node properties:

Procedure

1. Open the message flow that you want to configure.
2. Set the `Transaction mode` property for the input nodes in this message flow.

   The value that you set determines the behavior of the input node, and sets the default behavior for the rest of the message flow. Typically, you can choose the value **Yes** or **No**:

   - **Yes** means that the input node completes its own operation under sync point. The default behavior in the message flow is for actions to be taken under sync point.
   - **No** means that the input node completes its own operation out of sync point. The default behavior in the message flow is for actions to be taken out of sync point.

   Some nodes have extra or alternative values. For example, you can set the property on the MQInput node to **Automatic**, which means that the node gets the message under sync point if the message is persistent, and out of sync point if it is non-persistent.

   For details of the specific options for and actions that are taken by each node, see the relevant node reference topic. Review the node description, its properties, and the tabs on which the properties are set, because the resulting behavior is not identical across all input node types. Alternatively, you can read how to configure the following specific node types:

   - "Configuring JMS and SOAP nodes for local transactions" on page 700
   - "Configuring MQ nodes for transactions" on page 701
3. If your message flow includes nodes that interact with external resources, including output, request, and reply nodes, you can set a transaction property on most of these nodes.

   Set the property only if you want to change the behavior of the individual node from the default behavior for the message flow, which you set on the input node. The value that you set on this node has no effect on subsequent nodes in the message flow. If the node does not have a transactional property, its behavior is governed by the default behavior for the message flow, which you set in the input node.

   If your message flow is updating a database from multiple nodes in a single message flow, read "Message flow transactions" on page 691 to understand the possible interactions.

   a) Set the `Transaction` property for each node, if supported.

   b) Set the properties that define how errors are handled, if supported.
      For example, for nodes like the Compute node that can access databases, set the **Treat warnings as errors** and **Throw exception on database error** properties to define how that node handles database warnings and errors. Whether you select these properties, and how you connect the failure terminals of the nodes, also affect how database updates are committed or rolled back.

After you have configured your message flow, you must add it to a BAR file before you can deploy it. When you add it to a BAR file, the message flow is compiled, and more properties are available for configuration.

On distributed systems, use the `Coordinated Transaction` property to configure for globally coordinated transactions. By default, this property is cleared (not selected), which means the message flow is using local transactions, and the integration node commits or rolls back the message flow transaction. If you select this property, the transaction is globally coordinated; the input node calls the external transaction manager IBM MQ for commit and rollback processing. For more information, see

the “Coordinating transactions” on page 692 section in the topic “Message flow transactions” on page 691.

To configure message flow properties:

4. Add the message flow to a BAR file.
5. Select the **Manage and Configure** tab below the BAR file editor view, and select the message flow.

   The configurable properties for the message flow in the BAR file are displayed in the **Properties** view.

   Select `coordinatedTransaction` to configure the message flow as globally coordinated; when you set this property, the external transaction manager (IBM MQ) coordinates the transaction with all the resource managers that you have defined to the queue manager.

What to do next

When message flow design and development is complete, you can deploy the BAR file to the integration server on which you want the message flow to run.

If you are configuring your message flows for globally coordinated transactions, additional configuration is required. You, or your system administrator, must ensure that your App Connect Enterprise environment, the transaction manager, and the participating resource managers are all correctly configured to support coordinated transactions, before you run the message flow. For more information on what might be required, see “Configuring global coordination of transactions (two-phase commit)” on page 702.

If the App Connect Enterprise environment, the transaction manager, and the external resource managers are not correctly configured for global coordination, the message flow will run, but transactions will not be globally coordinated.

Configuring JMS and SOAP nodes for local transactions

When you include a node that uses JMS transport in a message flow, such as the JMSInput or SOAPInput node, the options that you select for the properties define behavior for the coordinated transaction.

Before you begin

Review “Configuring transactionality for message flows” on page 698 to understand what configuration is required for transactionality.

Procedure

- The option that you select for the `Transaction Mode` property defines whether the message is written under sync point:
  - If you set this property to `Yes` with the `Coordinated transaction` flow property selected, the message is received under external sync point coordination; that is, within an IBM MQ unit of work. Any messages that are sent later, by an output node in the same instance of the message flow, are put under sync point, unless the output node overrides this setting explicitly.
  - If you set this property to `Yes` with the `Coordinated transaction` flow property not selected, the message is received under the local sync point control of the node. Any messages that are sent later, by an output node in the flow, are not put under local sync point, unless an individual output node specifies that the message must be put under local sync point.
  - If you set this property to `No`, the message is not received under sync point. Any messages that are sent later, by an output node in the flow, are not put under sync point, unless an individual output node specifies that the message must be put under sync point.

What to do next

To receive messages under external sync point, you must take additional configuration steps, which need be applied only the first time that a specific node using JMS transport is deployed to the integration server for a particular JMS provider.

- On distributed systems, an IBM MQ queue manager provides the coordinated transaction support, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the message flow. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.
- Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use global (XA) coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions. For more information, see "Configuring JMS and SOAP nodes to support globally coordinated transactions" on page 721.

  For more information on transactionality, see Transactional support in the IBM MQ product documentation online.

Configuring MQ nodes for transactions
When you define an MQInput, MQGet, MQOutput, or MQReply node, the options that you select for the properties define behavior for the transaction.

Before you begin

Review "Configuring transactionality for message flows" on page 698 to understand what configuration is required for transactionality.

About this task

You can use IBM MQ resources in either local or globally coordinated transactions:

- For local transactions, you can have multiple queue managers participating in a message flow. Connections to queue managers can be local, client, or both.
- For globally coordinated transactions, you must specify a local queue manager on the integration node. This queue manager is the global transaction manager, and no other IBM MQ resources can be used in the message flow. For more information about configuring for globally coordinated transactions, see "Configuring global coordination of transactions (two-phase commit)" on page 702.

  If the connection is lost to the queue manager that was specified on the integration node in a globally-coordinated transaction, the integration node enters a standby state until the queue manager becomes available again. If the integration node was configured by using an MQ Service, the integration node is stopped.

If a connection to a queue manager is lost, and that connection is enlisted in a transaction, automatic reconnection is delayed until the inflight transaction is complete. Other queue managers that are required, but are not part of the transaction, reconnect automatically without delay. A completed transaction can include the following cases:

- The unavailable MQ resource was not required and was not used, because exception handling was defined in the message flow, so the inflight transaction was successful and committed.
- The unavailable MQ resource was required, and the message flow cannot succeed without it, so the inflight transaction was rolled back.

Review how to set the properties for MQ nodes to determine how they participate in transactions.

Configuring transactions for MQInput and MQGet nodes

Procedure

- The option that you select for the `Transaction Mode` property defines whether the message is read under sync point:
  - `Yes` (the default): if a transaction is not already inflight, the node starts a transaction.
  - `No`: the node does not start a transaction.
  - `Automatic`: if a transaction is not already inflight, and the message is specified in IBM MQ as persistent, the node starts a new transaction.

  All nodes that are subsequent in the message flow use the `Transaction mode` property that is set on the MQInput or MQGet node as their default transaction mode.
- If you set the `Browse Only` property for an MQInput or MQGet node, the value that is set for the `Transaction mode` property is ignored. However, when the `Transaction mode` property of the output node is set to **Automatic**, any messages that are propagated later to that output node in the same instance of the message flow use the `Transaction mode` value set on the input node.

  The `Browse Only` property for an MQGet node is on the **Request** tab.

Configuring transactions for MQOutput and MQReply nodes

Procedure

- The option that you select for the `Transaction Mode` property defines whether the message is written under sync point:
  - If you set the property to `Automatic`, the node participates in a transaction. If there is no inflight transaction, processed messages are committed immediately.
  - If you set the property to `Yes`, the node participates in the transaction currently inflight. If there is no transaction, a new transaction is started for this node, and all other nodes that are connected to its output terminal.
  - If you set the property to `No`, an inflight message is immediately committed.

What to do next

To use MQ nodes in global transactions, follow the steps in "Configuring global coordination of transactions (two-phase commit)" on page 702.

To understand when you might want to use a globally coordinated transaction for your message flow, see the "Coordinating transactions" on page 692 section in the topic "Message flow transactions" on page 691.

Configuring global coordination of transactions (two-phase commit)

Globally coordinate message flow transactions with a transaction manager to ensure the data integrity of transactions.

Before you begin

Read "Message flow transactions" on page 691 to understand how IBM App Connect Enterprise handles transactions. Depending on the external resource managers that are accessed during the processing of deployed message flows, you must complete the appropriate resource-dependent set of tasks to ensure that all resources are configured correctly. For example, you might have to create and configure databases, following the tasks in "Working with databases" on page 1130.

The IBM MQ queue manager that is associated with the integration node performs the transaction manager role, which means that App Connect Enterprise requires access to IBM MQ when processing messages. For more information, see "Enhanced flexibility in interactions with IBM MQ " on page 26.

You, or your message flow developer, must also ensure that the message flows deployed to the integration node are set up to support coordination. The tasks for configuring the message flows correctly are described in “Configuring transactionality for message flows” on page 698.

About this task

If you have configured an integration node that specifies a local MQ queue manager, you can set up globally coordinated transactions for the message flows that are managed by the integration node. The specified queue manager then performs the transaction manager role.

The following external resources are able to participate in a globally-coordinated message flow transaction:

- IBM MQ queues and messages
- CICS
- Databases
- JMS providers

On distributed platforms, the default behavior of the integration node is to manage all message flow transactions by using local transactions (a one-phase commit approach). In many contexts this approach is sufficient, but if your business requires assured data integrity and consistency (for example, for audit reasons, or for financial transactions), you can configure the integration node with a local IBM MQ queue manager to manage the message flow transactions as globally coordinated (a two-phase commit approach), by using the XA protocol standard. To globally coordinate transactions, there must be a queue manager specified on the integration node, and that queue manager performs the transaction manager role. The queue manager that is specified on the integration node is used as a globally coordinated resource. Other queue managers cannot participate in the message flow transaction.

If the connection is lost to an IBM MQ resource when the message flow is running, automatic reconnection is delayed until the inflight transaction is complete; see “Configuring MQ nodes for transactions” on page 701.

You configure the IBM MQ queue manager to act as the transaction manager by updating its `qm.ini` file, to add definitions of the additional resource managers with which you want IBM MQ to coordinate updates. Resource managers must register with the transaction manager, and there are two methods of registration:

- Static registration: The transaction manager involves all the resource managers in a transaction that are using static registration, even if a resource manager does not have a role to play in that transaction. Therefore all resource managers that use static registration must be available for the beginning of each globally coordinated transaction, even if they are not actively participating in that transaction, otherwise the transaction fails.
- Dynamic registration: Resource managers using dynamic registration must register with the transaction manager on a per transaction basis to get involved in a transaction (that is, if they are being updated as part of the globally coordinated transaction). A resource manager that uses dynamic registration does not need to be available for the beginning of each globally coordinated transaction.

Follow the instructions for the resource managers that are relevant in your environment and note whether they use static or dynamic registration:

- CICS
- ODBC connections to databases
- JDBC connections to databases
- JMS providers

Procedure

- You must specify a local queue manager on the integration node, and configured that queue manager for global transactions. You must use this queue manager only for your MQ nodes in your message flow.
- Set the node `Transaction mode` property to either `Automatic` or `Yes` for each node that is required in the global transaction.

  Review which `Transaction mode` option to set for each type of node in its node reference topic
- Configure the BAR file to enable global transactions:
  a) Below the BAR file editor view, select the **Manage and Configure** tab.
  b) In the **Properties** view, select `coordinatedTransaction`.

Results

When you have completed these steps, your message flows are processed by using global coordination, which is managed by the queue manager.

You must complete all of the steps, because if your resources are not correctly configured for global coordination the message flow will run, but transactions will not be globally coordinated.

Configuring databases for global coordination of transactions

If your message flow interacts with a database, and you want to globally coordinate the updates made to the database with other actions within the message flow, configure your databases for globally coordinated transactions.

About this task

If you have configured an integration node that specifies a local MQ queue manager, you can set up globally coordinated transactions for the message flows that are managed by the integration node. The specified queue manager then performs the transaction manager role.

If you restart a database while the integration node is still running, you must also restart the integration node. The integration node cannot detect that the database has stopped, and IBM MQ therefore retains its old connections to the database. When the database starts again, the integration node tries, and fails, to use these connections.

To configure databases for coordinated message flows, follow the instructions relevant to your database manager:

- DB2
- Oracle
- Sybase

Configuring DB2 for global coordination of transactions

Before you begin

You must complete these steps for databases that you connect to with an ODBC or a JDBC connection.

You must have database administrator (DBA) privileges to perform the following tasks.

About this task

To configure DB2 database instances for global coordination of transactions:

Procedure

1. Windows

   Windows systems only: for each 32-bit DB2 instance that is involved in the global coordination, run the following commands to set the Transaction Process Monitor name (TP_MON_NAME) to MQ:

   ```
   db2 update dbm cfg using TP_MON_NAME MQ
   db2stop
   db2start
   ```

2. If you are connecting an integration node on a distributed platform to a DB2 instance on z/OS, you must configure DB2 Connect to enable support for global coordination.

   Ensure that you have already configured a DB2 alias to represent the database by using DB2 Connect.

   Perform the following tasks on the system that hosts the integration node:

   a) Turn on the Connection Concentrator by configuring the DB2 database manager configuration parameters so that the value of the **MAX_CONNECTIONS** parameter is greater than the value of the **MAX_COORDAGENTS** parameter:

   ```
   db2 update dbm cfg using MAX_CONNECTIONS max_connections_value
   ```

   where *max_connections_value* is greater than the existing value of the **MAX_COORDAGENTS** parameter.

   b) Define the SPM name as the name of the system that hosts the integration node:

   ```
   db2 update dbm cfg using SPM_NAME host_name
   ```

   where *host_name* is the TCP/IP name of the system that hosts the integration node.

   c) Stop, then restart DB2 on the system that hosts the integration node to apply the changes:

   ```
   db2stop
   db2start
   ```

   DB2 Connect is now configured to enable global coordination of message flows that are deployed to the integration node (on a distributed platform) and that access DB2 on z/OS.

Results

The DB2 database instances are now configured for global coordination.

What to do next

See “Configuring ODBC connections for globally coordinated transactions” on page 708.

Configuring Oracle for global coordination of transactions

Before you begin

You must complete these steps for databases that you connect to with an ODBC connection only.

You must have database administrator (DBA) privileges to perform the following tasks.

About this task

To configure Oracle databases for global coordination of transactions:

Procedure

1. Ensure that the JAVA_XA package is present on the Oracle database by using, for example, the following Oracle SQLPLUS command:

   ```
   describe JAVA_XA;
   ```

   For more information, see the Oracle product documentation.
2. Ensure that the user ID that the integration node uses to access the database has the necessary Oracle privileges to access the DBA_PENDING_TRANSACTIONS view.

   You can grant the required access by using, for example, the following Oracle SQLPLUS command:

   ```
   grant select on DBA_PENDING_TRANSACTIONS to userid;
   ```

   If more than one user ID is involved (for example, if IBM App Connect Enterprise and IBM MQ run under different user IDs), you also need the privilege FORCE ANY TRANSACTION.

Results

The Oracle databases are now configured for global coordination.

What to do next

See “Configuring ODBC connections for globally coordinated transactions” on page 708.

Configuring Sybase for global coordination of transactions

Before you begin

You must complete these steps for databases that you connect to with an ODBC connection only.

You must have database administrator (DBA) privileges to perform the following tasks.

Procedure

- To configure Sybase databases for global coordination of transactions, ensure that the user ID that the integration node uses to access the database has been granted the Sybase role of `dtm_tm_role`.

Results

The Sybase databases are now configured for global coordination.

What to do next

See “Configuring ODBC connections for globally coordinated transactions” on page 708.

Configuring the CICSRequest node for global transactions on distributed systems

Configure the CICSRequest node for local and globally coordinated transactions for CICS on Linux, UNIX, and Windows.

Before you begin

Review CICSRequest node to understand what `Transaction mode` properties you can set on a CICSRequest node for transactions.

Linux UNIX Ensure that the user that is running the IBM MQ queue manager, typically mqm, is a member of the mqbrkrs group.

About this task

If you have configured an integration node that specifies a local MQ queue manager, you can set up globally coordinated transactions for the message flows that are managed by the integration node. The specified queue manager then performs the transaction manager role.

The queue manager provides the coordinated transaction support, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the message flow. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

To configure the CICSRequest node for globally coordinated transactions:

Procedure

1. Create a CICS Connection policy for the CICSRequest node (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) On the `Integration server used for XA recovery` property, specify the integration server for XA recovery, and which will own the connection. You cannot use any other integration server with the policy. When this property is set, the CICSRequest node is treated as a global resource, otherwise it participates as a non-global resource.
   b) On the `CICS server` property, you can choose either a direct connection or a connection through CICS Transaction Gateway.
   c) On the `Security identity (DSN)` property, set up a valid security ID to authenticate the connection to CICS (see "Security identities for the integration server connecting to external systems" on page 2582).

   Note: You should not use multiple CICS Connection policies that connect to the same CICS system in the same transaction or you might have problems with the overall transaction results.

   For more information, see CICS Connection policy (CICSConnection).
2. In the IBM App Connect Enterprise Toolkit, switch to the Integration Development perspective.
3. In the BAR file properties, set the `Coordinated Transaction` message flow property value to **yes**.
4. In the Message Flow editor, set the **Transaction mode** property to `Yes` or `Automatic` for each CICSRequest node that is required in the globally coordinated transaction.
5. Copy the switch file. The IIBXASwitch switch file is supplied by IBM App Connect Enterprise, and uses dynamic XA registration (see "Configuring databases for global coordination of transactions" on page 704):
   - On Windows:
     - If you are using 64-bit IBM MQ, copy the 64-bit switch file `IIBXASwitch.dll` from the IBM App Connect Enterprise installation directory (*`IIB_installation_directory`*`\bin\IIBXASwitch.dll`) to the `\exits64` subdirectory in the IBM MQ installation directory (*`MQ_installation_directory`*`\exits64\IIBXASwitch.dll`).
     - If you are using 32-bit IBM MQ, copy the 32-bit switch file (`IIBXASwitch32.dll`) from the IBM App Connect Enterprise installation directory (*`IIB_installation_directory`*`\bin\IIBXASwitch32.dll`) to the `\exits` subdirectory in the IBM MQ installation directory, and rename it to `IIBXASwitch.dll` (*`MQ_installation_directory`*`\exits\IIBXASwitch.dll`).
   - On Linux and UNIX systems, create a symbolic link to the switch file that is supplied in your *`install_dir`*`/server/lib` directory.

     install_dir is the directory to which you installed IBM App Connect Enterprise. The default location for this directory is *`install_dir`*`/ace-12.0.`*`n`*`.0/server` on Linux, or `/opt/IBM/`

`mqsi/12.0.n.0/server` on UNIX systems. The default directory includes the version, release, modification, and fix of the product, in the format v.r.m.f (version.release.modification.fix).

Set up links in the `/var/mqm/exits64` directory. The file name is `libIIBXASwitch.so`. For example:

```
ln -s install_dir/server/lib/libIIBXASwitch.so /var/mqm/exits64/libIIBXASwitch.so
```

6. Add a stanza to the queue manager `qm.ini` file for CICS, in the following format:

   On Linux and UNIX:

   ```
   XAResourceManager:
       Name=name
       SwitchFile=libIIBXASwitch.so
       XAOpenString=integrationNodeName,CICS,CICS_configurable_service
       XACloseString=
       ThreadOfControl=THREAD
   ```

   On Windows:

   ```
   XAResourceManager:
       Name=name
       SwitchFile=IIBXASwitch.dll
       XAOpenString=integrationNodeName,CICS,CICS_policy
       XACloseString=
       ThreadOfControl=THREAD
   ```

   where *name* is any name, *integrationNodeName* is the name of the integration node, and *CICS_policy* is the name of the CICS policy.

What to do next

- Check your configuration:
  - In the message flow, ensure that the `Coordinated Transaction` property is enabled for the BAR file by using the IBM App Connect Enterprise Archive editor.
  - Ensure that each node that is part of the globally-coordinated (XA) transaction has its `Transaction Mode` property set to a valid global transaction value (`Yes` or `Automatic`).
  - Ensure that the service ID that is used for the integration node and the queue manager is the same user ID.
  - Ensure that CICS two-phase commit is working by using the activity log. Define an Activity log policy by naming a file and setting the **RM** property to `CICS`.
  - Look for issues in the error log for the queue manager.
- Optional: secure the message flow connection; see "Creating a security profile" on page 2584.

Configuring ODBC connections for globally coordinated transactions
Configure the definition of your ODBC databases to the transaction manager (the queue manager).

Before you begin

You must create and configure the databases by following the instructions in "Working with databases" on page 1130.

You, or your message flow developer, must also ensure that the message flows deployed to the integration node are set up to support coordination. The tasks that are involved in configuring the message flows correctly are described in "Configuring transactionality for message flows" on page 698.

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with App Connect Enterprise, see "Installing IBM MQ" on page 96.

Procedure

1. Ensure that your databases are configured for global coordination.

   For the databases that your message flows connect to, complete the tasks that are described in “Configuring databases for global coordination of transactions” on page 704.
2. Configure the App Connect Enterprise environment so that the integration node queue manager coordinates transactions with database resource managers.

   The steps to configure the environment depend on the database manager that you are using.

   - “Configuring global coordination with DB2” on page 709
   - “Configuring global coordination with Oracle” on page 712
   - “Configuring global coordination with Sybase” on page 716

Results

When you have completed these steps, your message flows are processed by using global coordination, which is managed by the queue manager.

You must complete all the steps correctly; if you do not, global coordination will not work.

Configuring global coordination with DB2
Configure your IBM App Connect Enterprise environment to globally coordinate message flow transactions with updates in DB2 databases under the control of your integration node queue manager.

Before you begin

Ensure that the databases are configured for global coordination of transactions, see “Configuring databases for global coordination of transactions” on page 704.

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

Procedure

Follow the instructions appropriate to your platform:

- Linux “Linux and UNIX” on page 709
- Windows “Windows” on page 711

Linux and UNIX

Procedure

1. Run the **mqsimanagexalinks** command to create the links that are required by IBM MQ to include databases in XA transactions.

   For more information, see command.
2. Configure the integration node queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.
   a) Open the queue manager's `qm.ini` file in a text editor.

      The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`, where *queue_manager_name* is the name of the integration node that is associated with the queue manager.

   b) At the end of the `qm.ini` file, paste the following stanza:

```
XAResourceManager:
```

```
Name=DB2
SwitchFile=db2swit
XAOpenString=db=MyDataSource,uid=MyUserId,pwd=MyPassword,toc=t
XACloseString=
ThreadOfControl=THREAD
```

c) On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:

- *MyDataSource* is the name of the data source to which you want to connect.
- *MyUserId* must be the user name that the integration node uses to connect to the database.

  You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

  i) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

  ii) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

  iii) The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command
- *MyPassword* is the password that is associated with the user name.

d) Accept the default values for all the other lines in the stanza.
For example:

```
XAResourceManager:
Name=DB2
SwitchFile=db2swit
XAOpenString=db=MYDB,uid=wbrkuid,pwd=wbrkpw,toc=t
XACloseString=
ThreadOfControl=THREAD
```

3. The DB2 switch file is supplied by IBM App Connect Enterprise, and uses dynamic XA registration (see “Configuring databases for global coordination of transactions” on page 704).
4. Linux UNIX

   Stop then restart the queue manager to apply the changes, because `qm.ini` is read only while the queue manager is running. Before you restart the queue manager, ensure that the `ODBCINI` and `ODBCSYSINI` environment variables have been exported, and that the user ID that the queue manager processes will run under has read access to the `odbc.ini` and `odbcinst.ini` files.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/`*queue_manager_name*`/errors`, where *queue_manager_name* is the name of the queue manager that you restarted.

   When the queue manager restarts successfully, the changes that you made to `qm.ini` are applied.

Results

DB2 is now configured for global coordination with the integration node queue manager coordinating transactions.

What to do next

You can deploy globally coordinated message flows to the integration node.

Procedure

1. Windows

   Configure the integration node queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

   a) From the **Start** menu, open IBM MQ Explorer.

   b) Open the queue manager Properties dialog box, then open **XA resource managers**.

   c) Click the **Add** button to create a resource manager.

      i) In the **Name** field, enter a name to refer to a resource manager.

      ii) In the **SwitchFile** field, enter `db2swit`

      iii) Copy the provided DB2 switch files from the integration node install location to the queue managers exits and `exits64` directories (by default under `C:\Program Files (x86)\IBM\WebSphere MQ`).

      Copy the file:

      ```
      install_dir\server\sample\xatm\db2swit32.dll
      ```

      into the queue managers `exits` directory, and rename it to `db2swit.dll`.

      Copy the file

      ```
      install_dir\server\sample\xatm\db2swit.dll
      ```

      into the queue managers `exits64` directory.

      iv) In the **XAOpenString** field, paste the following string:

      ```
      db=MyDataSource,uid=MyUserId,pwd=MyPassword,toc=t
      ```

      v) In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

      a) *MyDataSource* is the name of the data source to which you want to connect.

      b) *MyUserId* must be the user name that the integration node uses to connect to the database.

      You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

      i. A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

      ii. A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

      iii. The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

      c) *MyPassword* is the password that is associated with the user name.

      For example:

      ```
      db=MYDB,uid=wbrkuid,pwd=wbrkpw,toc=t
      ```

      vi) Accept the default values for all the other fields on the page.

2. Windows

   Stop then restart the queue manager to apply the changes.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm -si queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in *install_dir*`\WebSphere MQ\Qmgrs\QMGR\errors`, where *install_dir* is the location in which the integration node is installed and QMGR is the name of your IBM MQ Queue manager.

   When the queue manager restarts successfully, the changes that you made are applied.

Results

DB2 is now configured for global coordination with the integration node queue manager coordinating transactions.

What to do next

you can deploy globally coordinated message flows to the integration node.

Configuring global coordination with Oracle
Configure your IBM App Connect Enterprise environment to globally coordinate message flow transactions with updates in Oracle databases under the control of an IBM MQ queue manager.

Before you begin

- Ensure that you have configured the databases for global coordination of transactions.

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

About this task

Configure your IBM App Connect Enterprise environment for global coordination by using a 64-bit queue manager as the transaction manager with the DataDirect drivers:

Procedure

1. Linux Windows

   Run the **mqsimanagexalinks** command.

2. Windows

   Add the directory specified as the queue manager’s *ExitsDefaultPath* to the system PATH.

3. Configure the integration node queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

   On Linux and UNIX:

   a) Open the queue managers' `qm.ini` file in a text editor.

   The `qm.ini` file is located at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`. Where *queue_manager_name* is the name of the integration node that is associated with the queue manager.

   b) Add the following stanza to the end of the `qm.ini` file:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc95.so
XAOpenString=ORACLE_XA
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +ServiceName=MyServiceName
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2
XACloseString=
ThreadOfControl=THREAD
```

   c) On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:

   - *MyHostName* is the name of the TCP/IP host that hosts the Oracle database listener. When you are using Oracle Real Application Clusters with multiple listeners for the given Service Name, if the Oracle listener identified by the values for *MyHostName* and *MyPortNumber* in the **XAOpenString** is unavailable, the alternative Oracle listeners that you might have defined in the **AlternateServers** list in your odbc.ini file are also tried.
   - *MyPortNumber* is the TCP/IP port on which the Oracle database listener is listening.
   - *MyUserId* must be the user name that the integration node uses to connect to the database.

     You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

     i) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

     ii) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

     iii) The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

   - *MyPassword* is the password that is associated with the user name.
   - *MyDataSourceName* is the ODBC data source name for the database, as defined in your odbc.ini file.
   - *MyServiceName* is the value set for the Service Name in the stanza for *MyDataSourceName* in your odbc.ini file.

   d) Accept the default values for all the other lines in the stanza.
   For example:

   - On AIX:

```
XAResourceManager:
Name=OracleXA
SwitchFile=UKoradtc95.so
XAOpenString=ORACLE_XA
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521
   +ServiceName=accounts_service
```

```
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+K=2
XACloseString=
ThreadOfControl=THREAD
```

e) If you are using Global coordination to Oracle:

- Set the ODBCINI environment variable to be visible in the environment from which you start your queue manager. The ODBCINI variable must reference the same file as the one being used by your integration node.
- An optional additional property, `CTO=`*`Value`*, is available for the **XAOpenString**. `CTO` is the value set for a Connection TimeOut, indicating the number of seconds that the Oracle XA switch file will wait for a response from the Oracle database to an XA request. For example, the timeout can be used to prevent long delays in the integration node failing over to an alternative Oracle Real Application Clusters node, when the active Oracle instance fails abruptly leaving socket connections hanging. The *Value* should be set to a value larger than the Oracle session timeout `sestm` value set in the **XAOpenString**. If this property is used, you must set the corresponding connection option. `ConnectionTimeout=`*`Value`* for the data source in the `odbc.ini` file. If this property is not used, or is set to zero, there will be no timeout (this is the default behavior).

On Windows:

a) From the **Start** menu, open IBM MQ Explorer.

b) Open the queue manager Properties dialog box, then open **XA resource managers**.

c) In the **SwitchFile** field, enter the name of the switch file, `ukora95.dll`.

d) In the **XAOpenString** field, paste the following string:

```
ORACLE_XA+
   +HostName=MyHostName
   +PortNumber=MyPortNumber
   +ServiceName=MyServiceName
   +ACC=P/MyUserId/MyPassword
   +sestm=100+threads=TRUE
   +DataSource=MyDataSourceName
   +K=2
```

e) In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

- *MyHostName* is the name of the TCP/IP host that hosts the Oracle database listener. When you are using Oracle Real Application Clusters with multiple listeners for the given Service Name, if the Oracle listener identified by the values for *MyHostName* and *MyPortNumber* in the

XAOpenString is unavailable, the alternative Oracle listeners that you might have defined in the **AlternateServers** list in your odbc.ini file are also tried.

- *MyPortNumber* is the TCP/IP port on which the Oracle database listener is listening.
- *MyUserId* must be the user name that the integration node uses to connect to the database.

  You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

  i) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

  ii) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

  iii) The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command
- *MyPassword* is the password that is associated with the user name.
- *MyDataSourceName* is the ODBC data source name for the database, as defined in your odbc.ini file.
- *MyServiceName* is the value set for the Service Name in the ODBC definition for the data source *MyDataSourceName*.

For example:

```
ORACLE_XA+
   +HostName=diaz.hursley.ibm.com
   +PortNumber=1521
   +ServiceName=accounts_service
   +ACC=P/wbrkuid/wbrkpw
   +sestm=100+threads=TRUE
   +DataSource=MYDB+K=2
```

f) Accept the default values for all the other fields on the page.

4. The Oracle switch file is supplied by IBM App Connect Enterprise, and uses static XA registration by default. On AIX and Linux on x86-64 only, if you want to enable Oracle data sources to perform dynamic XA registration, set the following environment variable: DDTEK_XA_DYNAMIC_REGISTRATION=1 (See "Configuring databases for global coordination of transactions" on page 704).
5. Linux UNIX

   Stop then restart the queue manager to apply the changes, because qm.ini is read only while the queue manager is running. Before you restart the queue manager, ensure that the ODBCINI and ODBCSYSINI environment variables have been exported, and that the user ID that the queue manager processes will run under has read access to the odbc.ini and odbcinst.ini files.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in /var/mqm/qmgrs/*queue_manager_name*/errors, where *queue_manager_name* is the name of the queue manager that you restarted.

   When the queue manager restarts successfully, the changes that you made to qm.ini are applied.

6. Windows

   Stop then restart the queue manager to apply the changes.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm -si queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/`*queue_manager_name*`/errors`, where *queue_manager_name* is the name of the queue manager that you restarted.

   When the queue manager restarts successfully, the changes that you made are applied.

Results

Oracle is now configured for global coordination with the integration node queue manager coordinating transactions.

What to do next

You can deploy globally coordinated message flows to the integration node.

Configuring global coordination with Sybase

Configure you IBM App Connect Enterprise environment to globally coordinate message flow transactions with updates in Sybase databases under the control of a queue manager.

Before you begin

- Ensure that the databases are configured for global coordination of transactions.

On distributed systems, an IBM MQ queue manager that is associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

About this task

To configure your IBM App Connect Enterprise environment for global coordination by using an IBM MQ queue manager as the transaction manager with the DataDirect drivers:

Procedure

1. Run the **mqsimanagexalinks** command.
2. Configure the integration node queue manager with XA resource manager information for each database that is involved in the transaction that the queue manager will globally coordinate.

   Linux **On Linux and UNIX:**

   a) Open the queue manager's `qm.ini` file in a text editor.

   The `qm.ini` file is at `/var/mqm/qmgrs/`*queue_manager_name*`/qm.ini`, where *queue_manager_name* is the name of the integration node that is associated with the queue manager.

   b) At the end of the `qm.ini` file, paste the following stanza:

   ```
   XAResourceManager:
       Name=SYBASEXA
       SwitchFile=UKasedtc95.so
       XAOpenString=-NSYBASEDB -AMyServerName,MyPortNumber -Uuid -Ppwd -K2
       XACloseString=
   ```

```
    ThreadOfControl=THREAD
```

c) On the **XAOpenString** line, replace the following values with values that are appropriate for your configuration:

- *MyServerName* is the name of the TCP/IP host that hosts the Sybase ASE server.
- *MyPortNumber* is the TCP/IP port on which the Sybase ASE server is listening.
- *uid* must be the user name that the integration node uses to connect to the database.

  You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

  i) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

  ii) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

  iii) The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

- *pwd* is the password that is associated with the user name.

d) Accept the default values for all the other lines in the stanza.
For example:

- On AIX:

```
XAResourceManager:
    Name=SYBASEXA
    SwitchFile=UKasedtc95.so
    XAOpenString=-NSYBASEDB -Adiaz,4100 -Uwbrkuid -Pwbrkpw -K2
    XACloseString=
```

```
ThreadOfControl=THREAD
```

On Windows:

a) From the **Start** menu, open IBM MQ Explorer.

b) Open the Properties dialog box, then open **XA resource managers**.

c) In the **SwitchFile** field, enter the name of the switch file, `ukase95.dll`.

d) In the **XAOpenString** field, paste the following string:

```
-NSYBASEDB -AMyServerName,MyPortNumber -WWinsock -Uuid -Ppwd -K2
```

e) In the **XAOpenString** field, replace the values with values that are appropriate for your configuration:

- *install_dir* is the location in which IBM App Connect Enterprise is installed.
- *MyServerName* is the name of the TCP/IP host that hosts the Sybase ASE server.
- *MyPortNumber* is the TCP/IP port on which the Sybase ASE server is listening.
- *uid* must be the user name that the integration node uses to connect to the database.

  You can define the user name that the integration node uses in a number of ways; make sure that you specify the correct name in this file. The integration node determines the user name by checking the following conditions in the order listed:

  i) A specific user name and password for this data source name (DSN), that you have defined by running the **mqsisetdbparms** command.

  ii) A default user name and password for all DSNs, that you have defined by running the **mqsisetdbparms** command.

  iii) The integration node service user name, which you define with the **-i** parameter on the **mqsicreatebroker** command

- *pwd* is the password that is associated with the user name.

For example:

```
-NSYBASEDB -Adiaz,4100 -WWinsock -Uwbrkuid -Pwbrkpw -K2
```

f) Accept the default values for all the other fields on the page.

3. The Sybase switch file is supplied by IBM App Connect Enterprise, and uses static XA registration (see “Configuring databases for global coordination of transactions” on page 704).
4. Linux UNIX

Stop then restart the queue manager to apply the changes, because `qm.ini` is read only while the queue manager is running. Before you restart the queue manager, ensure that the `ODBCINI` and `ODBCSYSINI` environment variables have been exported, and that the user ID that the queue manager processes will run under has read access to the `odbc.ini` and `odbcinst.ini` files.

To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

```
endmqm queue_manager_name
strmqm queue_manager_name
```

When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in `/var/mqm/qmgrs/`*queue_manager_name*`/errors`, where *queue_manager_name* is the name of the queue manager that you restarted.

When the queue manager restarts successfully, the changes that you made to `qm.ini` are applied.

5. Windows

   Stop then restart the queue manager to apply the changes.

   To stop and restart the queue manager, enter the following commands, where *queue_manager_name* is the name of the queue manager:

   ```
   endmqm queue_manager_name
   strmqm -si queue_manager_name
   ```

   When the queue manager restarts, check the queue manager log for all warnings that are associated with the restart. The log files are located in *install_dir*\WebSphere MQ\Qmgrs\QMGR\errors, where *install_dir* is the location in which the integration node is installed and QMGR is the name of your IBM MQ Queue manager.

   When the queue manager restarts successfully, the changes that you made are applied.

Results

Sybase is now configured for global coordination with your queue manager coordinating the transactions.

What to do next

You can deploy globally coordinated message flows to the integration node.

Configuring a JDBC type 4 connection for globally coordinated transactions
If you want the database that you access through a JDBC type 4 connection to participate in globally coordinated transactions, set up the appropriate environment.

Before you begin

Set up your JDBC provider definition.

About this task

On distributed systems, an IBM MQ queue manager associated with the integration node performs the transaction manager role, which means that IBM App Connect Enterprise requires access to IBM MQ when processing messages. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

Updates that you make to a database across a JDBC type 4 connection can be coordinated with other actions taken within the message flow, if you set up the resources to support coordination.

Complete the following steps:

Procedure

1. Check that the definition of your JDBC Providers policy is appropriate for coordinated transactions.

   For example, to set up the required JDBC classes:

   - For DB2, set **JDBC type 4 data source class name** to `com.ibm.db2.jcc.DB2XADataSource` and **JDBC driver class name** to `com.ibm.db2.jcc.DB2Driver`.
   - For Oracle, set **JDBC type 4 data source class name** to `oracle.jdbc.xa.client.OracleXADataSource` and **JDBC driver class name** to `oracle.jdbc.OracleDriver`.

   Consult your database administrator or the documentation provided by your database supplier, to confirm that all the JDBC Providers policy properties are set appropriately. For example, a database supplier might require secure access if it is participating in coordinated transactions.

2. Define the switch file and the database properties:
   a) The JDBC switch file is supplied by IBM App Connect Enterprise, and uses static XA registration (see “Configuring databases for global coordination of transactions” on page 704).
   b) Linux

      On Linux and UNIX systems, open the `qm.ini` file for the integration node queue manager with a text editor. Add the following stanza for each database:

      ```
      XAResourceManager:
             Name=Database_Name
             SwitchFile=JDBCSwitch
             XAOpenString=JDBC_DataSource
             ThreadOfControl=THREAD
      ```

      Database_Name is the database name (DSN) of the database defined to the JDBC Providers policy.

      `JDBCSwitch` is a fixed generic name that represents the switch file for XA coordination. Use this value, or another single fixed value, in each stanza; the specific switch file that the queue manager uses is defined by the symbolic links you create in the next step.

      JDBC_DataSource is the identifier of the JDBC Providers policy.

      Define a stanza for each database (DSN) that you connect to from this integration server. You must create separate definitions even if the DSNs resolve to the same physical database. Therefore, you must have a stanza for each JDBC Providers policy that you have defined, because each service can define the properties for a single database.
   c) Windows

      On Windows, open IBM MQ Explorer and select the queue manager, for example BROKERQM.

      Open the **XA resource manager** page, and modify the attributes to create the definition of the database. The attributes are the same as those shown for Linux and UNIX; **Name**, **SwitchFile**, **XAOpenString**, and **ThreadofControl**. Leave the additional attribute, **XACloseString**, blank.

      Enter *JDBCSwitch* in **SwitchFile**.
3. Set up queue manager access to the switch file:
   a) Linux

      On Linux and UNIX systems, create a symbolic link to the switch files that are supplied in your *install_dir*`/server/lib` directory.

      install_dir is the directory to which you installed IBM App Connect Enterprise. The default location for this directory is *install_dir*`/ace-12.0.`*n*`.0/server` on Linux, or `/opt/IBM/mqsi/12.0.`*n*`.0/server` on UNIX systems. The default directory includes the version, release, modification, and fix of the product, in the format v.r.m.f (version.release.modification.fix).

      Set up links in the `/var/mqm/exits64` directory. The file name for all platforms is `libJDBCSwitch.so`.

      Specify the same name of the switch file, `JDBCSwitch` or your own value, in the `/exits64` directory. For example, on AIX:

      ```
      ln -s install_dir/server/lib/libJDBCSwitch.so /var/mqm/exits64/JDBCSwitch
      ```
   b) Windows

      On Windows, copy the `JDBCSwitch32.dll` file from the *install_dir*`\server\bin` directory to the `\exits` subdirectory in the IBM MQ installation directory, and rename the file to `JDBCSwitch.dll`. Then, copy the `JDBCSwitch.dll` file from the *install_dir*`\server\bin` directory to the `\exits64` subdirectory in the IBM MQ installation directory.

4. Configure the message flow that includes one or more nodes that access databases that are to participate in a globally coordinated transaction.
   a) Open an IBM App Connect Enterprise Toolkit session.
   b) Switch to the Integration Development perspective.
   c) Add the message flow that includes the node or nodes that connect to the database that is to participate in a globally coordinated transaction to a new or existing BAR file.
   d) Build the BAR file.
   e) Click the **Configure** tab, select the message flow that you have added, and select the **Coordinated Transaction** check box.

What to do next

If your integration node is running on Windows, authorize the queue manager to access resources that are associated with the JDBC Providers policy (see "Authorizing access to JDBC type 4 JDBCProvider resources on Windows" on page 204).

If you have been following the instructions in "Working with databases" on page 1130, the next task is "Configuring ODBC connections for globally coordinated transactions" on page 708 (optional).

Configuring JMS and SOAP nodes to support globally coordinated transactions

To include message flow nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

Before you begin

Review "Configuring JMS and SOAP nodes for local transactions" on page 700 to understand what properties you must set on a SOAP or JMS node to use globally coordinated transactions.

Note: Global (XA) transaction coordination for nodes that use JMS transport is supported only if the message flow that contains the nodes is deployed to an integration node that has a dedicated queue manager; the integration node cannot share a queue manager with other integration nodes.

About this task

If you require global transaction coordination, choose a JMS provider that conforms to the Java Message Service Specification version 1.1 or 2.0, and that supports the JMS XAResource API through the JMS session.

If you specify your own JMS provider by using the JMS Providers policy, set the `XA is supported` property of the policy to true to indicate that the selected JMS provider supports XA coordinated transactions. If you set this property to true, and the selected JMS provider does not support XA transactions, an exception is raised. If you set this property to false, but the `Transaction mode` property on the node is set to `Yes` and the `Coordinated Transaction` message flow property is selected, an exception is raised.

If the message designer specified a non-XA-compliant provider, only the non-transactional mode is supported. In this case, you must set the `Transaction mode` property to `None` for all JMS and SOAP nodes that use JMS transport.

On distributed systems, an IBM MQ queue manager provides the coordinated transaction support, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the flow. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

Procedure

To configure the message flow nodes, complete the following steps:

1. In the IBM App Connect Enterprise Toolkit, switch to the Integration Development perspective.
2. In the BAR file properties, set the `Coordinated Transaction` message flow property value to **yes**.

3. In the Message Flow editor, set the **Transaction mode** property to Yes for each node that uses JMS transport that is required in the XA coordinated transaction.
4. Create a Queue Connection Factory, and use either the default name, *recoverXAQCF*, or supply your own name.

   For more information about creating JNDI administered objects, see "JNDI administered objects" on page 866.
5. There are two modes for enlisting an XAResource with the transaction coordinator:

   a. Static enlistment, which means that the XAResource is called to perform a start transaction when any message flow in the same Integration Bus starts a global coordinated transaction.

   b. Dynamic enlistment where the JMS node registers an interest in joining a global transaction when it has work to perform.

   Before deployment on distributed systems, you must set up a stanza for each JMS provider that you want to use. For Windows, the JMS provider switch file is called `DynJMSSwitch.dll`. For all other operating systems, the JMS provider switch file is called `libDynJMSSwitch.so`.

   Follow the steps in the appropriate topic for the operating system, or systems, that your enterprise uses:

   - Linux Linux and UNIX systems
   - Windows Windows systems

     On Windows only, you must also modify the queue manager authorization, as described in "Windows systems: modifying the queue manager authorization" on page 882.

What to do next

The JMS provider might supply additional JAR files that are required for transactional support; for more information, see the documentation that is supplied with the JMS provider. For example, on distributed systems, the IBM MQ JMS provider supplies an extra JAR file, `com.ibm.mqetclient.jar`.

You must add any additional JAR files to the `shared_classes` directory:

- Linux On Linux and UNIX: `var/mqsi/shared-classes`.
- Windows On Windows: `C:\ProgramData\IBM\MQSI\shared-classes`.

For more information, see the section about making the JMS provider client available to the JMS nodes in node.

Configuring the queue manager to coordinate JMS resources on Linux and UNIX

Configure the global (XA) resource managers for the queue manager by defining a stanza in the integration node's queue manager `qm.ini` file for each new JMS provider.

About this task

The JMS provider can be specified by a JMS or SOAP node that is included in a message flow that is running on the integration node.

Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use XA coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.

To configure the queue manager for Linux and UNIX:

Procedure

1. Add a stanza to the queue manager `.ini` file for each JMS provider, in the following format:

```
XAResourceManager:
    Name=<Name>
```

```
SwitchFile=/install_dir/server/lib/libDynJMSSwitch.so
XAOpenString=<Initial Context Factory>,
          <location of JNDI bindings>'
          <LDAP Principal>,
          <LDAP Credentials>,
          <Recovery Connection Factory Name>,
          <JMS Principal>,
          <JMS Credentials>
          ThreadOfControl=THREAD
```

The parameters for the stanza are as follows:

<Name>
Required. The name of the defined JMS provider resource manager.

<SwitchFile>
Required. The file system path to the Switch library that is supplied in the `bin` directory of the integration node. There are two options:

a. The library called JMSSwitch, which manages XAResources that statically enlist with the transaction coordinator.

b. The library called DynJMSSwitch, which manages dynamically enlisted XAresources.

install_dir
The location of the IBM App Connect Enterprise installation. This value is required where the LDAP parameters are omitted, but a user-defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
Required. The Initial Context Factory identifier for the JMS provider. This value is set in the JMSInput node property `Initial context factory`.

<Location of JNDI bindings>
Required. Use either the filepath to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply a file path to the bindings file, do not include the file name. For more information about creating the JNDI administered objections, see node or node.

The values for the *Initial Context factory* and *Location of JNDI bindings* in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

<LDAP Principal>
Optional. Specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects. `LDAP Principal` must match the value that is set for the integration node by using the **`mqsicreatebroker`** command. `LDAP Principal` is not currently supported, it is reserved for future use.

<LDAP Credentials>
Optional. Specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects. `LDAP Credentials` must match the value that is set for the integration node by using the **`mqsicreatebroker`** command. `LDAP Credentials` is not currently supported, it is reserved for future use.

<Recovery Connection Factory Name>
Optional. Specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes. If a value is not specified, you must add a default value for `recover`XAQCF to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA

Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

<JMS Principal>
Optional. Provide the user ID required to connect to a JMS provider, by using a secure JMS Connection Factory. `JMS Principal` is not currently supported, it is reserved for future use.

<JMS Credentials>
Optional. Provide the password that is required to connect to the same JMS provider in conjunction with the JMS principal. `JMS Credentials` is not currently supported, it is reserved for future use.

The JMS Principal and JMS Credentials must be configured together.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma, as seen in the following example stanza.

The example stanza describes IBM MQ Java as the JMS provider for global transactions:

```
XAResourceManager:
    Name=XAJMS_PROVIDER1
    SwitchFile=/install_dir/server/lib/libDynJMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

- `XAJMS_PROVIDER1` is the user-defined name for the resource manager
- `/install_dir/server/lib/libDynJMSSwitch.so` is the installation path
- `com.sun.jndi.fscontext.RefFSContextFactory` is the *<Initial Context Factory>*
- `/Bindings/JMSProvider1_Bindings_Directory` is the location of the bindings file
- `myJMSuser1` is the *<JMS Principal>*
- `passwd` is the password used in *<JMS Credentials>*

In this example, the optional fields `<LDAP Principal>`, `<LDAP Credentials>`, and `<Recovery Connection Factory Name>` are not required, therefore the positional comma delimiters only are configured in the `XAOpenString` stanza.

2. Update the Java PATH environment variable for the queue manager of the integration node to point to the `bin` directory in which the switch file is located.
For example:

```
install_dir/server/bin
```

Switch files are installed in the *install_dir*`/server/lib` directory. To simplify the contents of the `qm.ini` file, create a symbolic link to the switch file for the queue manager to retrieve from the `/var/mqm/exits64` directory. For example:

```
ln -s install_dir/server/lib/libDynJMSSwitch.so /var/mqm/exits64/libDynJMSSwitch
```

What to do next

Check your configuration:

- In the message flow, ensure that the coordinated property is enabled by using the IBM App Connect Enterprise Archive editor.
- Ensure that each message flow node that is part of the XA transaction is set to the global transaction mode.

- Ensure that the service ID that is used for the integration node and the queue manager is the same user ID.
- Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings by using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see Transactional support in the IBM MQ product documentation online.

Configuring the queue manager to coordinate JMS resources on Windows
Use IBM MQ Explorer to configure the global (XA) resource managers for the queue manager.

About this task

Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use XA coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.

To configure the queue manager for Windows:

Procedure

1. Open IBM MQ Explorer.
2. Select the queue manager for your integration node and click **Properties**.
3. Select **XA resource managers** in the left pane and click **Add**.
4. Complete the fields to define a new resource manager:
   a) **Name**: Enter the name of the resource manager; for example, `IIBWMQJMS`.
   b) **SwitchFile**: Enter **DynJMSSwitch**.
   c) On **XAOpenString**, enter the following values, which are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

      Initial Context Factory
      The Initial Context Factory identifier for the JMS provider; this value is required.

      Location of JNDI bindings
      Either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply the file path to the bindings file, do not include the file name. See the JMSInput or

JMSOutput node for further details about creating the JNDI administered objects; this value is required.

The values for the *Initial Context factory* and *Location of JNDI binding*s in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

LDAP Principal
Optional: The principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects. `LDAP Principal` is not currently supported, it is reserved for future use.

LDAP Credentials
Optional: The credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

All LDAP parameters must match the values that you specified on the **mqsicreatebroker** command. `LDAP Credentials` is not currently supported, it is reserved for future use.

Recovery Connection Factory Name
Optional: The name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non-default name is required.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory that is called `recover`XAQCF is used. In either case, this value must refer to a JNDI administered object that has already been created.

JMS Principal
The user ID that is required to connect to a JMS provider, using a secure JMS Connection Factory. `JMS Principal` is not currently supported, it is reserved for future use.

JMS Credentials
The password that is required to connect to the same JMS provider with the JMS principal. `JMS Credentials` is not currently supported, it is reserved for future use.

The JMS Principal and JMS Credentials must be configured together.

a) **XACloseString**: Leave this field blank.

b) **ThreadOfControl**: Set the value to `Thread`.

5. Click **OK** to complete the XA resource manager definition.
6. Click **OK** to close the queue manager properties dialog.
7. Click **File** > **Exit** to close IBM MQ Explorer.
8. Copy the `DynJMSSwitch.dll` file to the `\exits64` folder in the IBM MQ installation directory.
9. Copy the `DynJMSSwitch32.dll` file to the `\exits` directory in the IBM MQ installation directory and then rename the file to `DynJMSSwitch.dll`.
10. Set the `XAOpenString` property to a string value as follows: *`Initial Context,location JNDI,Optional_parms`*.
11. Set the `ThreadOfControl` property to `Thread`.

What to do next

- Check your configuration:
  - In the message flow, ensure that the coordinated property is enabled by using the IBM App Connect Enterprise Archive editor.
  - Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
  - Ensure that the service ID that is used for the integration node and the queue manager is the same user ID.
  - Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
    - If you create the bindings by using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.
- Authorize the integration node and queue manager to access shared resources that are associated with the JMS provider; see "Windows systems: modifying the queue manager authorization" on page 882.

For more information, see Transactional support in the IBM MQ product documentation online.

Troubleshooting

IBM App Connect Enterprise examines the IBM MQ registry to determine if the queue manager is XA enabled. If IBM App Connect Enterprise is unable to locate the location of the queue managers working directory, an error is issued at startup. You can specify this location by setting the following environment variable in the `mqsiprofile` script: `MQ_DATA_PATH` For example:

```
MQ_DATA_PATH="C:\Program Files (x86)\IBM\IBM MQ"
```

Configuring message flows for data conversion

If you exchange messages between applications that run on systems that are incompatible in some way, you can configure your system to provide data conversion as the message passes through the integration node.

About this task

Data conversion might be necessary if either of the following two values are different on the sending and receiving systems:

1. CCSID. The Coded Character Set Identifier refers to a set of coded characters and their code point assignments. IBM App Connect Enterprise can process and construct application messages in any code page for which IBM MQ provides conversion to and from Unicode, on all operating systems. For more information about code page support, see the *Application Programming Reference* section of the IBM MQ product documentation online.

   This behavior might be affected by the use of other products in conjunction with IBM App Connect Enterprise. Check the documentation for other products, including any databases that you use, for further code page support information.
2. Encoding. This setting defines the way in which a machine encodes numbers; that is, binary integers, packed-decimal integers, and floating point numbers. Numbers that are represented as characters are handled in the same way as all other string data.

If the native CCSID and encoding on the sending and receiving systems are the same, you do not need to call data conversion processes.

IBM App Connect Enterprise and IBM MQ provide data conversion facilities to support message exchange between unlike systems. Your choice of which facilities to use depends on the characteristics of the messages that are processed by your message flows:

- Messages that contain text only
- Message that include numerics

- Messages that are self-defining

Messages that contain text only

Read this section if your messages are IBM MQ messages that contain all text (character data or string).

If IBM MQ supports the systems on which both sending and receiving applications are running for data conversion, use IBM MQ facilities which provide the most efficient data conversion option.

The default behavior of IBM MQ is to put messages to queues specifying the local system CCSID and encoding. Applications issuing MQGET can request that the queue manager provides conversion to their local CCSID and encoding as part of get processing.

To use this option:

1. Design messages to be text-only. If you are using COBOL, move numeric fields to USAGE DISPLAY to put them into string form.
2. Set the Format field in the MQMD to MQFMT_STRING (value MQSTR).
3. Call MQGET with MQGMO_CONVERT in the receiving application. If you prefer, you can convert when the message is received by the integration node, by setting the Convert property of the MQInput node to yes (by selecting the check box).

If you require more sophisticated data conversion than IBM MQ provides in this way (for example, to an unsupported code page), use IBM MQ data conversion exits. For more information about these, see the *Application Programming Reference* section of the IBM MQ product documentation online.

Messages that include numerics

Read this section if your messages include numeric data, or are text only but are not IBM MQ messages.

If these messages can be predefined (that is, their content and structure is known and predictable), use the facilities provided by IBM App Connect Enterprise and the MRM.

All application messages are handled by the integration node in Unicode, to which they are converted on input, and from which they are converted on output. You can configure message flows to influence the way in which output messages are constructed.

To use this option:

1. Define the output message in the MRM domain. You can create this definition in one of the following ways:
   - Import an external message definition (for example a C header or COBOL copybook).
   - Create the message model in the message definition editor.
2. Configure a message flow to receive and process this message:
   a. If you include an MQInput node, do not request conversion by this node.
   b. Include a Compute node in the message flow to create the output message with the required content:
      - If the output message is an IBM MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the MQMD.

        For example, to set values for a target z/OS system running with CCSID of 37 and encoding of 785:

        ```
        SET OutputRoot.MQMD.CodedCharSetId = 37;
        SET OutputRoot.MQMD.Encoding = 785;
        ```

      - If the output message is not an IBM MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the Properties folder.

Messages that are self-defining

Read this section if your messages are self-defining.

Self-defining messages are supported in the XML and JMS domains. These messages are all text and can be handled by IBM MQ, if they originate from, or are destined for, IBM MQ applications. If not, use IBM App Connect Enterprise facilities by setting the CCSID and Encoding fields in the Properties folder in the message when it passes through a Compute node.

Defining a promoted property

When you create a message flow, you can promote properties from individual nodes in that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes.

Before you begin

Read the concept topic about promoted properties.

About this task

You can perform the following tasks related to promoting properties:

- "Promoting a property" on page 729
- "Renaming a promoted property" on page 732
- "Removing a promoted property" on page 733
- "Converging multiple properties" on page 735

Some of the properties that you can promote to the message flow are also configurable; you can modify them when you deploy the BAR file in which you have stored the message flow to each integration node. If you set values for configurable properties when you deploy a BAR file, the values that you set override values set in the individual nodes, and those that you have promoted.

Promoting a property

You can promote a message flow node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple message flow nodes in the flow by converging promoted properties.

Before you begin

- Create a message flow, as described by "Creating a message flow" on page 574
- Read the concept topic about promoted properties

About this task

The majority of message flow node properties are available for promotion, but you cannot promote the following properties:

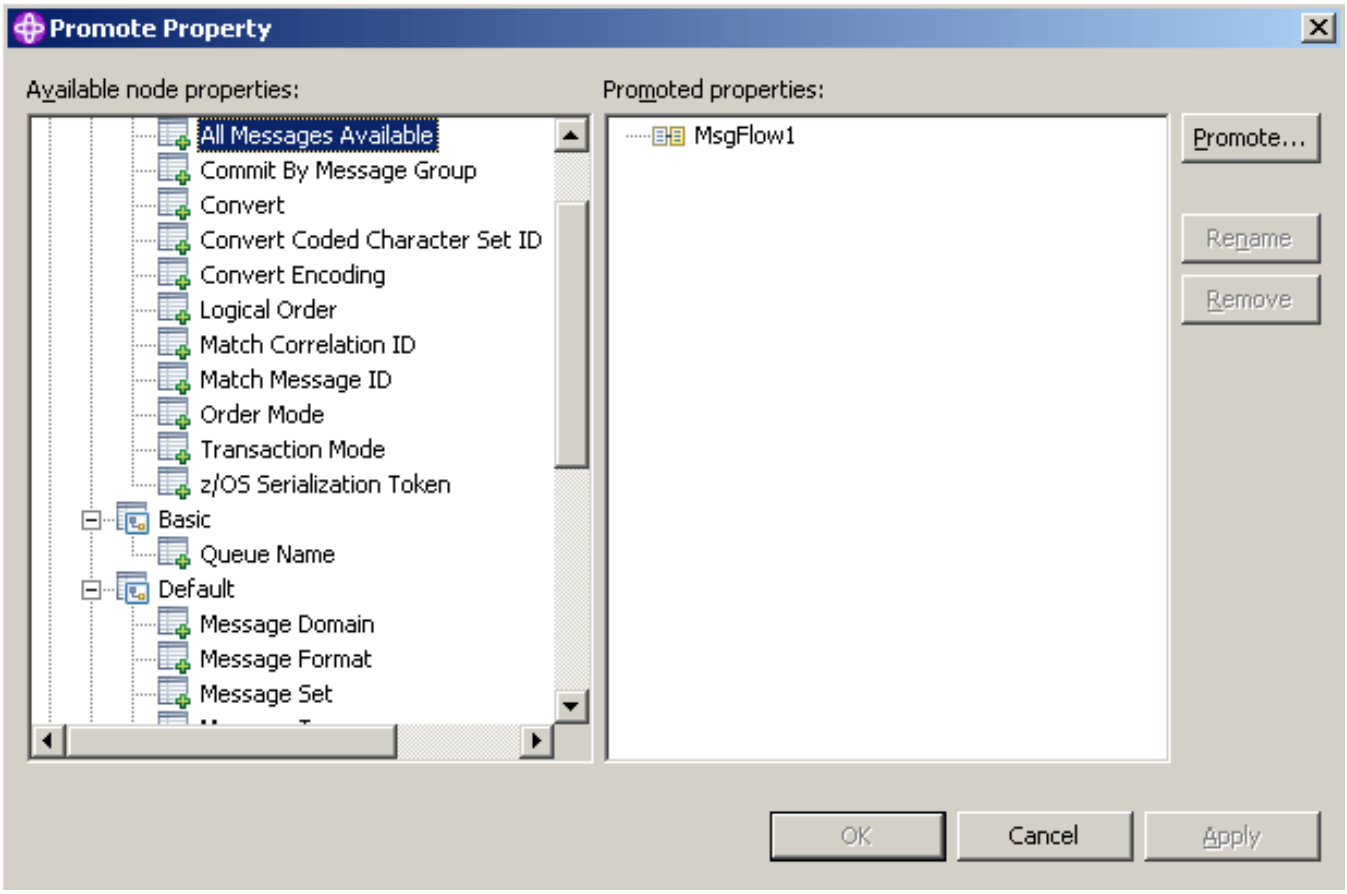
- Properties that name mapping modules
- A property group (but you can promote an individual property)
- A property that you cannot edit (for example, the `Fix` property of the MQInput node)
- The description properties (`Short Description` and `Long Description`)
- Complex properties (for example, the `Query elements` property of the DatabaseRoute node, or the `Opaque elements` property of the MQInput and several other nodes)

Procedure

To promote message flow node properties to the message flow level, complete the following steps:

1. Switch to the Integration Development perspective.
2. Open the message flow for which you want to promote properties.
3. Right-click the appropriate node and click **Promote Property**.

   The Promote Property dialog box is displayed.



   The Available node properties pane lists all available properties for all the nodes in the message flow. The properties for the node that you clicked are expanded. You can expand the properties for all the nodes in the open message flow, regardless of the node that you clicked initially.

   The Promoted properties pane displays the name of the open message flow and all the properties that are currently promoted to the message flow. If you have not yet promoted any properties, only the message flow name is displayed as the root of the promoted property tree, as shown in the previous example. If you have already promoted properties from this node, the properties appear in the Promoted properties pane, but not in the Available node properties pane.

4. Select the property or properties that you want to promote to the message flow.

   You can select multiple properties by holding down **Ctrl** and selecting the properties.

5. Click **Promote**.

   The **Target Selection** dialog box opens and displays valid targets for the promotion.

6. Select the destination group or property for the properties that you want to promote. You can group together related properties from the same or different nodes in the message flow by dropping the selected properties onto a group or property that already exists, or you can create a new target for the promotion by clicking **New Group** or **New Property**.

   You can rename groups and properties by selecting them and clicking **Rename**.

7. Click **OK** to confirm your selections and close the **Target Selection** dialog box.

   If you create a new group or property by using the **Target Selection** dialog box, the changes persist even if you select **Cancel** in the dialog box. When the dialog box closes, groups or properties that you have created by using the **Target Selection** dialog box appear in the **Promote Property** dialog box.

You can remove any of these properties from the **Promote Property** dialog box by selecting them and clicking **Remove**.

8. Click **OK** to commit your changes and close the **Promoted Property** dialog box.

   If you click **Apply**, the changes are committed but the dialog box remains open.

Results

The message flow node properties are promoted to the message flow. When you have promoted a property, you can no longer make any changes to that property at the node level; you can update its value only at the message flow level. To view the message flow's properties, click the message flow (not the individual nodes) in the Message Flow editor to display the properties in the Properties view. The properties that you have promoted are organized in the groups that you created. If you now set a value for one of these properties, that value appears as the default value for the property whenever the message flow is included in other message flows.

When you select an embedded message flow in another message flow (a subflow) and view its properties, you see the promoted property values. If you look inside the embedded flow (by selecting **Open Subflow**), you see the original values for the properties. The value of a promoted property does not replace the original property, but it takes precedence when you deploy the message flow.

Promoting properties by dragging

About this task

You can also promote properties from the **Promote Property** dialog box by dragging the selected property or properties from the Available node properties pane of the **Promote Property** dialog box to the Promoted properties pane, as described in the following steps.

Procedure

1. Select the property that you want to promote. You can select multiple properties by holding down **Ctrl**, and selecting the properties.
2. You can promote the selected properties using the following methods:
   - Drop the selected property or properties in an empty space.

     A new group is created automatically for the message flow, and the property is placed in it, with the original name of the property and the name of the message flow node from which it came displayed beneath the property entry.

     The name of the first group that is created is `Group1` by default. If a group called `Group1` already exists, the group is given the name `Group2`, and so on. You can rename the group by double-clicking it and entering new text, or by selecting the group in the Promoted properties pane and clicking **Rename**.

     When you create a new promoted property, the name that you enter is the name by which the property is known within the system, and must meet certain Java and XML naming restrictions. These restrictions are enforced by the dialog box, and a message is displayed if you enter a name that includes a non-valid character. For example, you cannot include a space or quotation marks (").

     If you are developing a message flow in a user-defined project that will be delivered as an Eclipse plug-in, you can add translations for the promoted properties that you have added. Translated names can contain characters, such as space, that are restricted for system names. The option to

provide translated strings for promoted properties is not available if you are working with a message flow in an integration project.

- Drop the selected property or properties onto a group that already exists, to group together related properties from the same or different nodes in the message flow.

  For example, you might want to group all promoted properties that relate to database interactions. You can change the groups to which promoted properties belong at any time by selecting a property in the Promoted properties pane and dragging it onto a different group.

- Drop the selected property or properties onto a property that already exists, to converge related properties from the same or different nodes in the message flow.

  For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

  For more information on converging properties, see "Converging multiple properties" on page 735.

Promoting mandatory properties

About this task

If you promote a property that is mandatory (that is, an asterisk appears beside the name in the Properties view), the mandatory characteristic of the property is preserved. When a mandatory property is promoted, its value does not need to be set at the node level. If the flow that contains the mandatory promoted property is included as a subflow in another flow, the property must be populated for the subflow node.

Promoting properties through a hierarchy of message flows

About this task

You can repeat the process of promoting message flow node properties through several levels of message flow. You can promote properties from any level in the hierarchy to the next level above, and so on through the hierarchy to the top level. The value of a property is propagated from the highest point in the hierarchy at which it is set down to the original message flow node when the message flow is deployed to an integration node. The value of that property on the original message flow node is overridden.

Renaming a promoted property
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

Before you begin

- Promote a property
- Read the concept topic about promoted properties

About this task

Procedure

To rename a promoted property:

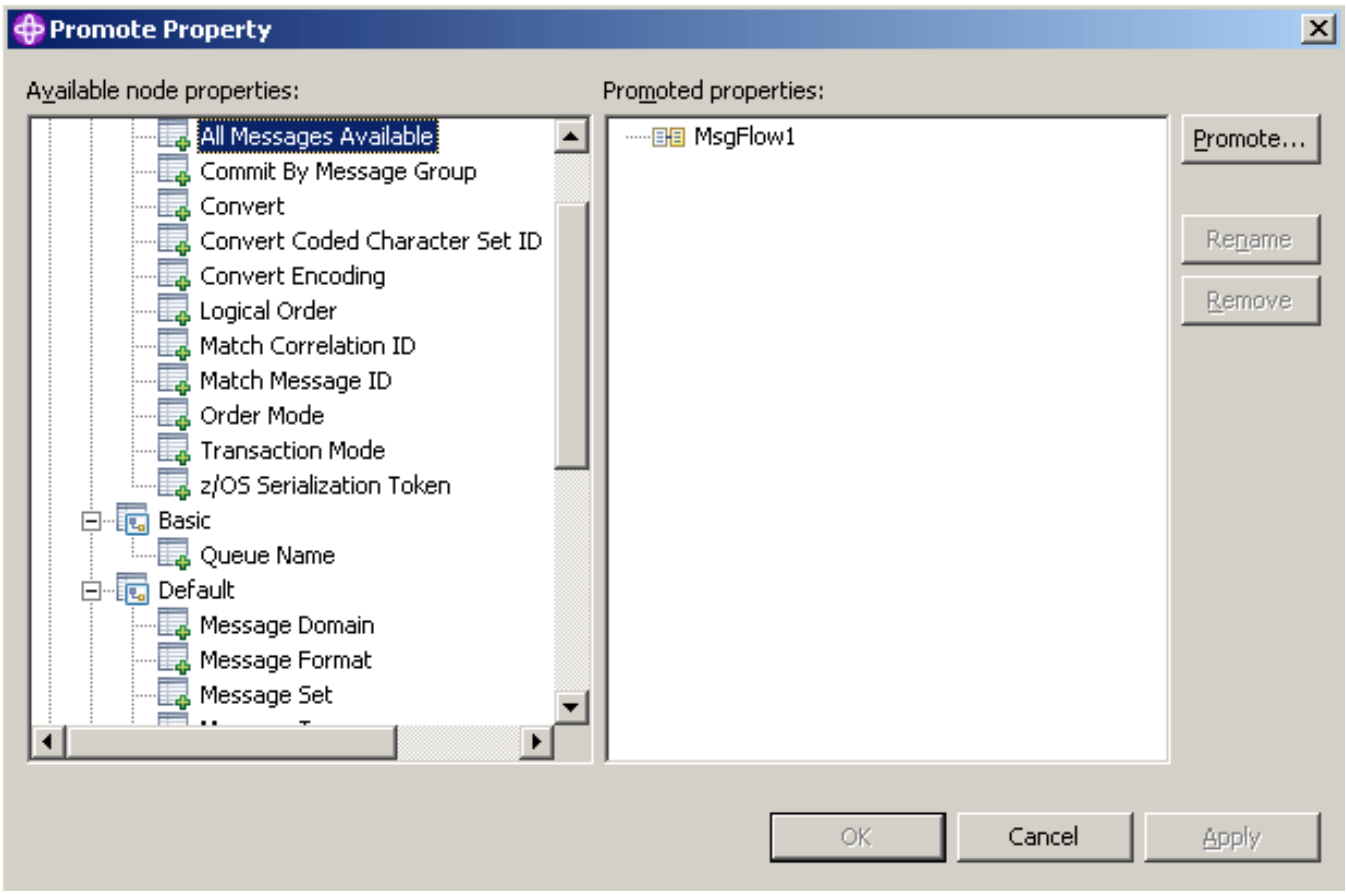
1. Switch to the Integration Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Application Development view.

   You can also open the message flow by right-clicking it in the Application Development view and clicking **Open**

   The message flow contents are displayed in the editor view.

3. In the editor view, right-click the symbol of the message flow node for which you want to promote properties.
4. Select **Promote Property**.

   The **Promote Property** dialog box is displayed.



5. Promoted properties are shown in the Promoted properties pane on the right of the **Promote Property** dialog box. Double-click the promoted property in the list of properties that are currently promoted to the message flow level, or select the property that you want to rename and click **Rename**.

   The name is highlighted, and you can edit it. Modify the existing text or enter new text to give the property a new name, and press Enter.
6. Click **Apply** to commit this change without closing the **Promote Property** dialog box. Click **OK** to complete your updates and close the dialog box.

Removing a promoted property
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a BAR file that includes this message flow.

Before you begin

- Promote a property
- Read the concept topic about promoted properties

About this task

Procedure

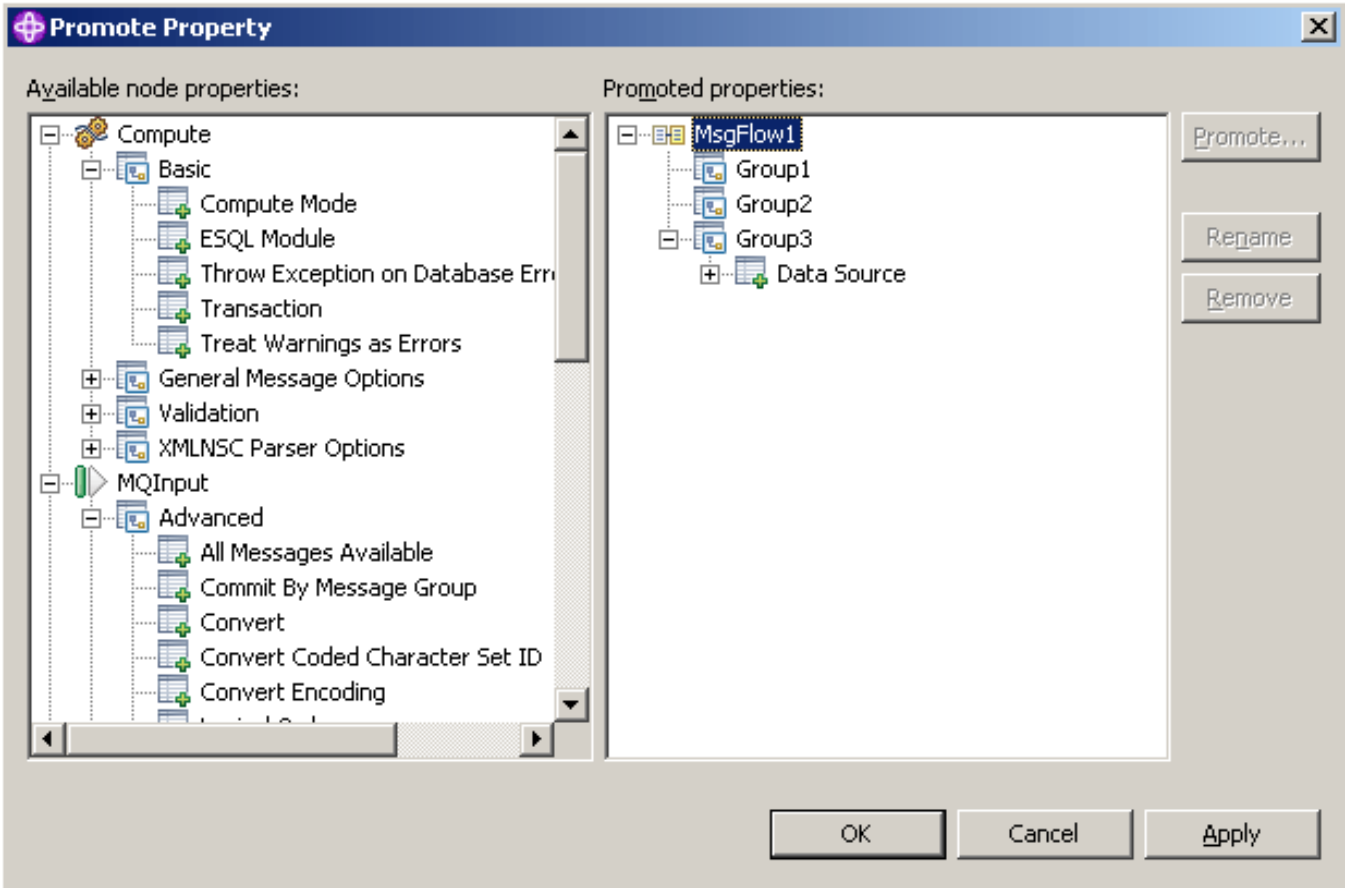
If you have promoted one or more message flow node properties, and want to delete them, complete the following steps:

1. Switch to the Integration Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Application Development view.

   You can also open the message flow by right-clicking it in the Application Development view and clicking **Open**

   The message flow contents are displayed in the editor view.
3. In the Editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog is displayed.



5. Select the promoted property that you want to remove from the list of properties in the Promoted properties pane, and click **Remove**.

   The property is removed from the Promoted properties pane and is restored to the list in the Available node properties pane, in the appropriate place in the tree of properties for the node from which you promoted it. You can promote this property again.
6. To delete all the promoted properties in a single group, select the group in the Promoted properties pane and click **Remove**.

   The group and all the properties it contains are deleted from this list: the individual properties that you promoted are restored to the nodes from which you promoted them.
7. Click **Apply** to commit this change without closing the **Promote Property** dialog box. Click **OK** to complete your updates and close the dialog box.

Results

If you have included this message flow in a higher-level message flow, and have set a value for a promoted property that you have now deleted, the embedding flow is not automatically updated to reflect the deletion. However, when you deploy that embedding message flow in the integration node domain, the deleted property is ignored.

Converging multiple properties
You can promote properties from several nodes in a message flow to define a single promoted property, which applies to all those nodes.

Before you begin

- Create a message flow, as described in "Creating a message flow" on page 574
- Read the concept topic about promoted properties

About this task

One example for the use of promoting properties is for database access. If a message flow contains two Database nodes that each refer to the same physical database, you can define the physical database just once on the message flow by promoting the `Data Source` property of each Database node to the message flow, and setting the property at the message flow (promoted) level.

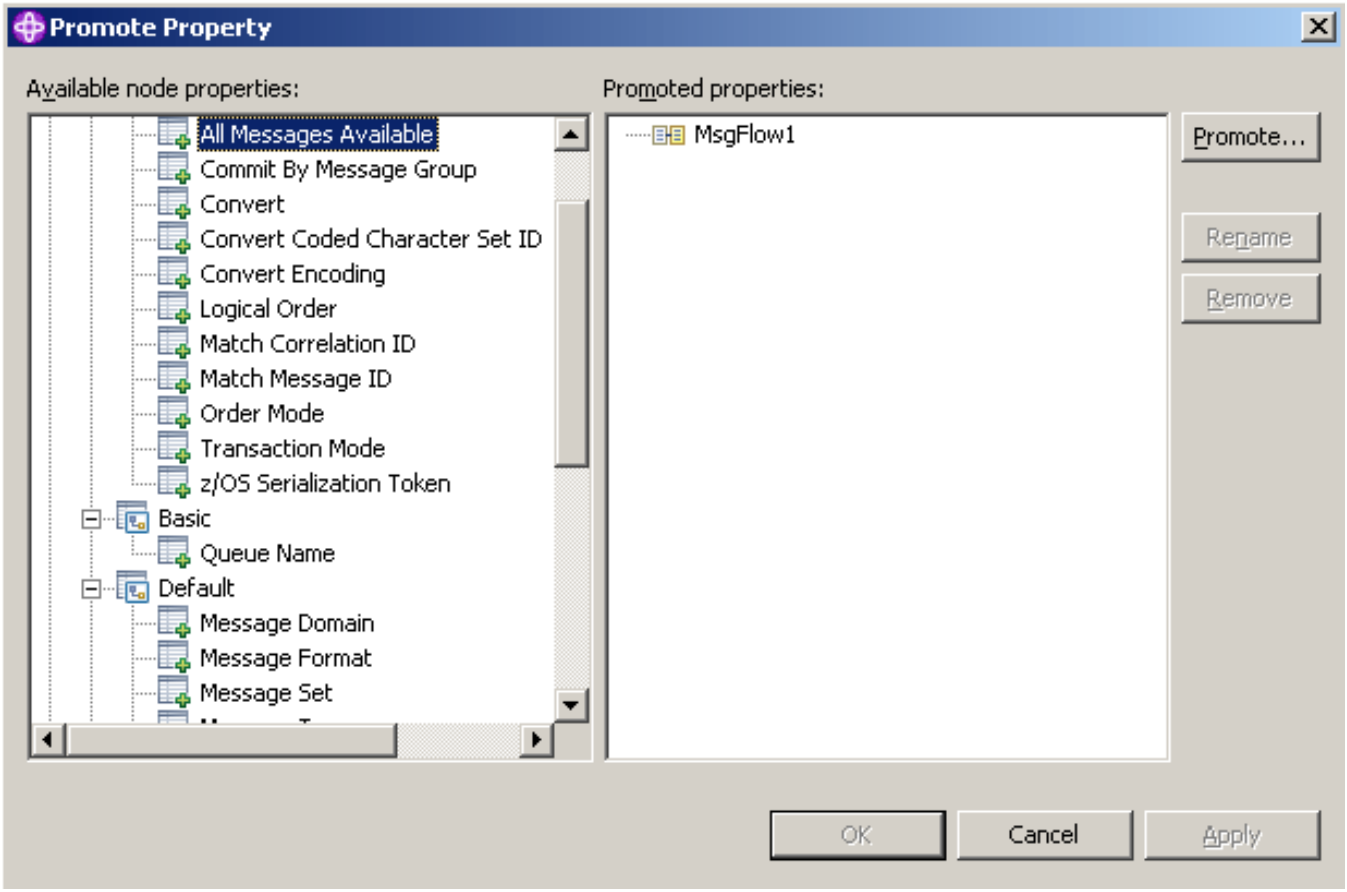
To converge multiple node properties to a single promoted property:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow in the Message Flow editor.

3. Right-click the node for which you want to promote the properties, then click **Promote Property**.

   The Promote Property dialog box is displayed.



4. Select the property that you want to converge.

   The list in the Available node properties pane initially shows the expanded list of all available properties for the selected node. If you have already promoted properties from this node, they do not appear in this pane, but they appear in the Promoted properties pane.

   The Available node properties pane also includes the other nodes in the open message flow. You can expand the properties that are listed under each node and work with all these properties at the same time. You do not have to close the dialog box and select another node from the Message Flow editor to continue promoting properties.

   You can select multiple properties to promote by selecting a property, holding down **Ctrl**, and selecting one or more other properties.

   If you have you selected multiple properties to converge, all the properties that you have selected must be available for promotion. If one or more of the selected properties is not available for promotion, the entire selection becomes unavailable for promotion, and the **Promote** button is disabled.

5. Click **Promote** to promote the property or properties

   The **Target Selection** dialog box opens:

   The **Target Selection** dialog box displays only the valid targets for the promotion of the previously selected property or properties and allows you to create a new target for the promotion, such as to a new group or to a new property.

6. To converge properties from the same or different nodes in the message flow, expand the tree and click a property that already exists.

   You can rename the properties by selecting them and clicking **Rename**, or by double-clicking the group or property.

7. Click **OK** to confirm your selections.

   Note: If you create a new group or property by using the **Target Selection** dialog box, the changes persist even if you click **Cancel**. When the dialog box closes, groups or properties that you have created by using the **Target Selection** dialog box appear in the **Promote Property** dialog box.
8. Expand the property trees for all the nodes for which you want to promote properties.
9. Drag the first instance of the property that you want to converge from the Available node properties pane, and drop it onto the appropriate group in the Promoted properties pane.
   - If the group already contains one or more promoted properties, the new property is added at the end of the group. You can rename the new property by double-clicking the property, or by selecting the property and clicking **Rename**.
   - If you want the promoted property to appear in a new group, drag the property into an empty space below the existing groups to create a new group. Alternatively:
     a. Select the property that you want to promote, and click **Promote**. The **Target Selection** dialog box opens.
     b. Click **New Group**, and enter the name of the new group.
     c. Click **OK** to confirm your changes.
   - If you drag the property onto an existing promoted property of a different type, a no-entry icon is displayed and you cannot drop the property. You must create this property as a new promoted property, or drop it onto a compatible existing promoted property. Properties must be associated with the same property editor to be compatible. For example, if you are using built-in nodes, you can converge only properties of the same type (string with string, Boolean with Boolean).

   If you are using user-defined nodes, you must check the compatibility of the property editors for the properties that you want to converge. If you have written compiler classes for a node, you must also ensure that converged properties have the same compiler class.
10. Drag all remaining instances of the property from each of the nodes in the Available node properties pane onto the existing promoted property. The new property is added under the existing promoted property, and is not created as a new promoted property.
11. Click **Apply** to commit this change without closing the **Promote Property** dialog box. Click **OK** to complete your updates and close the dialog box.

    You can also converge properties from the **Promote Property** dialog box by dragging the selected property or properties from the Available node properties pane to the Promoted properties pane:
    a. Select the property that you want to promote. You can select multiple properties to promote by selecting a property, holding down **Ctrl**, and selecting one or more other properties.
    b. Drop the selected property or properties onto a property in the Promoted properties pane to converge related properties from the same or different nodes in the message flow.

       For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

Results

You have promoted properties from several nodes to define a single promoted property, which is used for all those nodes.

Connecting client applications

Connect your client applications to the integration node by using one or more of the supported protocols from other resources and software servers in your network.

About this task

- "Processing IBM MQ messages" on page 738

- “Processing HTTP messages” on page 787
- “Processing web service messages” on page 805
- “Working with JMS” on page 870
- “Processing MQTT messages” on page 887
- “Processing Kafka messages” on page 892
- “Processing TCP/IP messages” on page 911
- “Processing email messages” on page 938
- “Working with files” on page 951
- “Connecting to external REST APIs” on page 1032
- “Connecting to an App Connect REST API” on page 1047
- “Connecting to Enterprise Information Systems” on page 1049
- “Working with databases” on page 1130
- “Working with IMS” on page 1145
- “Working with CORBA” on page 1152
- “Working with Salesforce” on page 1195
- “Working with LoopBack connectors” on page 1212

Processing IBM MQ messages

IBM App Connect Enterprise provides a number of nodes for handling messages that are received from or sent to IBM MQ applications.

About this task

If you intend to use IBM App Connect Enterprise to process MQ messages, you must install IBM MQ in addition to installing IBM App Connect Enterprise. On distributed systems, if you want to connect to a remote queue manager to process MQ messages, you must install either an MQ Client or MQ Server on the same machine as IBM App Connect Enterprise, in addition to installing MQ server on the machine that is running your queue manager. IBM MQ is available as a separate installation package, and your IBM App Connect Enterprise license entitles you to install and use IBM MQ with IBM App Connect Enterprise. For more information, see “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “Installing IBM MQ” on page 96.

The following nodes are provided for working with IBM MQ messages:

node
: Use an MQInput node if the messages arrive on an IBM MQ queue, and the node is to be at the start of a message flow.

node
: Use an MQOutput node if the target application expects to receive messages on an IBM MQ queue, or on the IBM MQ reply-to queue that is specified in the input message MD.

node
: Use an MQGet node to retrieve a message from an IBM MQ queue, if you want to get the message later in the message flow.

node
: Use an MQReply node if the target application expects to receive messages on the IBM MQ reply-to queue that is specified in the input message MD.

node
: Use the MQHeader nodes to manipulate IBM MQ transport headers and their properties without writing compute nodes. Use the MQHeader node to add, modify, or delete Message Descriptor (MD) and Dead Letter Header (DLH) headers.

node
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the IBM MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

node
Use the MQJMSTransform node to receive messages that have an IBM MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

The following topics contain information that you need to understand before you can use the IBM MQ support in IBM App Connect Enterprise:

- "Enhanced flexibility in interactions with IBM MQ " on page 26
- "IBM MQ topologies" on page 739
- "Configuring connections to IBM MQ" on page 743

For step-by-step information about how to use MQ nodes to perform a set of typical tasks, see "Working with IBM MQ" on page 753.

See the following topics for more information about configuring your message flows, and about related resources for handling messages that are received from or sent to IBM MQ applications:

- "Enabling IBM MQ applications" on page 777
- "Application programming interfaces" on page 779
- "Using IBM MQ cluster queues for input and output" on page 780
- "Using queues on an IBM MQ uniform cluster" on page 781
- "Configuring flows to handle IBM MQ message groups" on page 783
- "Ensuring that messages are not lost" on page 786
- connections

IBM MQ topologies

You can use IBM MQ to create flexible connection topologies from the different connectivity options.

If you already have an IBM MQ solution, you do not need to change your existing IBM MQ topology to work with IBM App Connect Enterprise; the products integrate automatically.

Your IBM MQ applications exchange messages and other data by communicating with message flows that are running in the integration server. You can connect your applications to the integration server by using one of the supported communication methods. If your applications are written to use IBM MQ, the requirement for the channels or client connections are determined by the types of nodes that you include in your message flows. These resources are application-specific, and you must create these resources yourself.

You might need some or all of the following resources:

- A queue manager that is associated with the integration node or server. For some capabilities, you might also need to define queues for the queue manager; see "IBM App Connect Enterprise features requiring supplementary products" on page 9.
- Defined TCP/IP listener connections on the queue manager that is associated with the integration node or server.

For more information about creating resources, see Intercommunication in the IBM MQ product documentation online.

Local and client connections

A local connection is a connection to a local queue manager that uses server bindings. A client connection connects to remote queue managers. You can use local connections, client connections, or a combination of local and client connections to your queue managers.

When you configure a connection from an MQ node to an IBM MQ queue manager, you can optionally configure the connection to use a security identity for authentication, SSL for confidentiality, or both. The security identity, which passes user name and password security credentials to the queue manager, can be used on connections to local or remote queue managers. For connections to remote queue managers, you can choose whether to use the SSL protocol to provide confidentiality on the client connection. IBM App Connect Enterprise supports a subset of the SSL functionality that is supported by IBM MQ. For more information, see "Connecting to a secured IBM MQ queue manager" on page 747.

For globally coordinated transactions, you must designate a local queue manager as the transaction manager by associating it with the integration node. Other queue managers can only participate as locally-coordinated in that message flow. For more information, see "Configuring MQ nodes for transactions" on page 701.

There are some extra considerations if you use a remote queue manager: for example, you might want to set up security for connecting over the network. For more information, see "Administration security overview" on page 2497.

Connection management

If a connection is lost between an integration server and queue manager, the integration server automatically attempts to reconnect to the queue manager. All resources and transports that do not require access to that connection continue to run. In a non-transactional message flow, MQInput, MQOutput, MQGet, and MQReply nodes automatically attempt to reconnect once. If the node still cannot connect, then normal exception processing occurs.

If a connection to a queue manager is lost, and that connection is enlisted in a transaction, automatic reconnection is delayed until the inflight transaction is complete. Other queue managers that are required, but are not part of the transaction, reconnect automatically without delay. A completed transaction can include the following cases:

- The unavailable MQ resource was not required and was not used, because exception handling was defined in the message flow, so the inflight transaction was successful and committed.
- The unavailable MQ resource was required, and the message flow cannot succeed without it, so the inflight transaction was rolled back.

To learn more about using IBM MQ in transactions, see "Configuring MQ nodes for transactions" on page 701.

Remote default connection management

If an integration server is configured to use a remote default queue manager connection, the server shuts down if the connection is lost. This is the same behavior as for a local default queue manager, and ensures that state information stored on queues is kept intact. Separately-configured client links, where a specific input node is connected to a different remote queue manager, follow the reconnect pattern described above, but all other MQ-related nodes that are using the remote default queue manager (including event-driven processing nodes) trigger a server shutdown if the connection is lost. For information about using a remote default queue manager connection, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

MQEndpoint policy

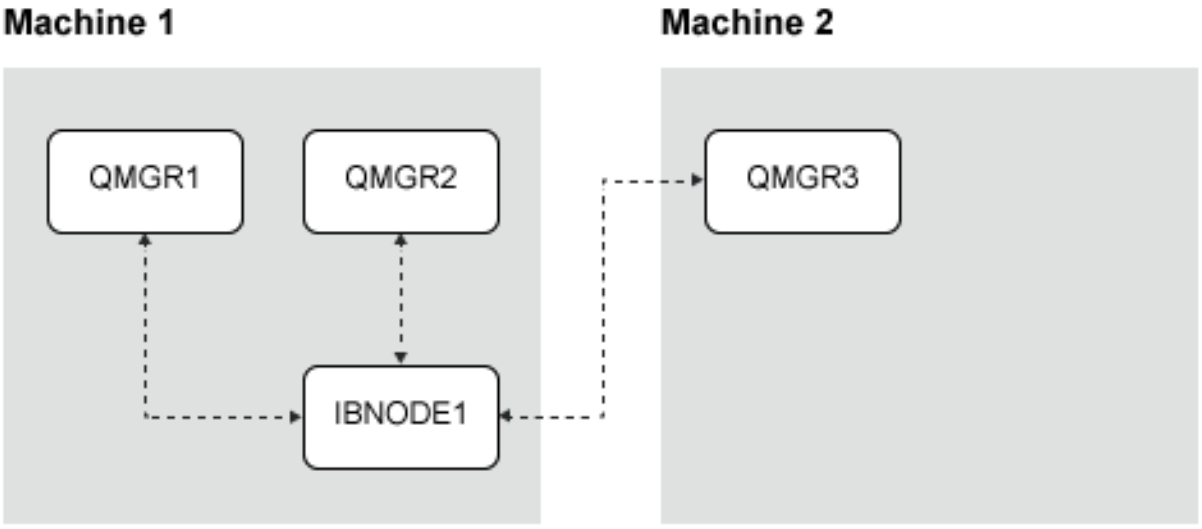
Connection details for MQ nodes can be managed by MQEndpoint policies (see policy).

IBM MQ topologies

You can have almost any topology configuration to connect between IBM App Connect Enterprise and IBM MQ by using combinations of multiple local and client connections. The following diagrams provide examples of these basic topologies.

Scenario 1

This scenario uses multiple queue managers to a single integration node.



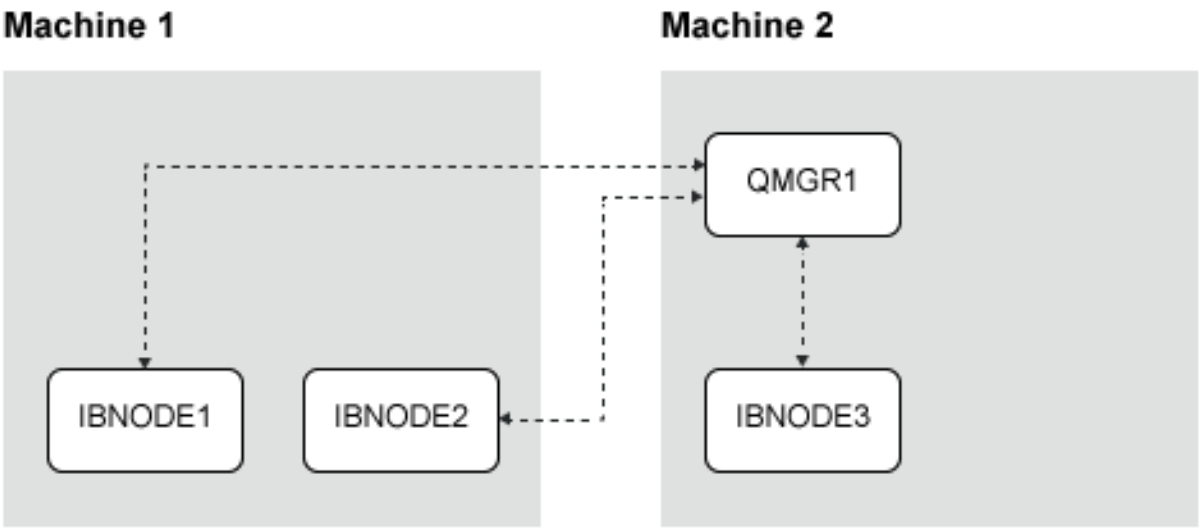
In this diagram, multiple queue managers are connected to a single integration node, IBNODE1. For example, the message flow might contain an MQInput and MQOutput node, which both share QMGR1 to process the messages. QMGR3 is available to IBNODE1 by a TCP/IP client connection.

Table 13. Strengths and weaknesses of scenario 1:

Strengths	**Weaknesses**
• The integration node can connect to a queue manager, both local and remote, without changing an existing IBM MQ topology	• Failover: there is a single point of failure (IBNODE1). To mitigate this risk, you can add more integration nodes

Scenario 2

This scenario uses multiple integration nodes to a single queue manager.



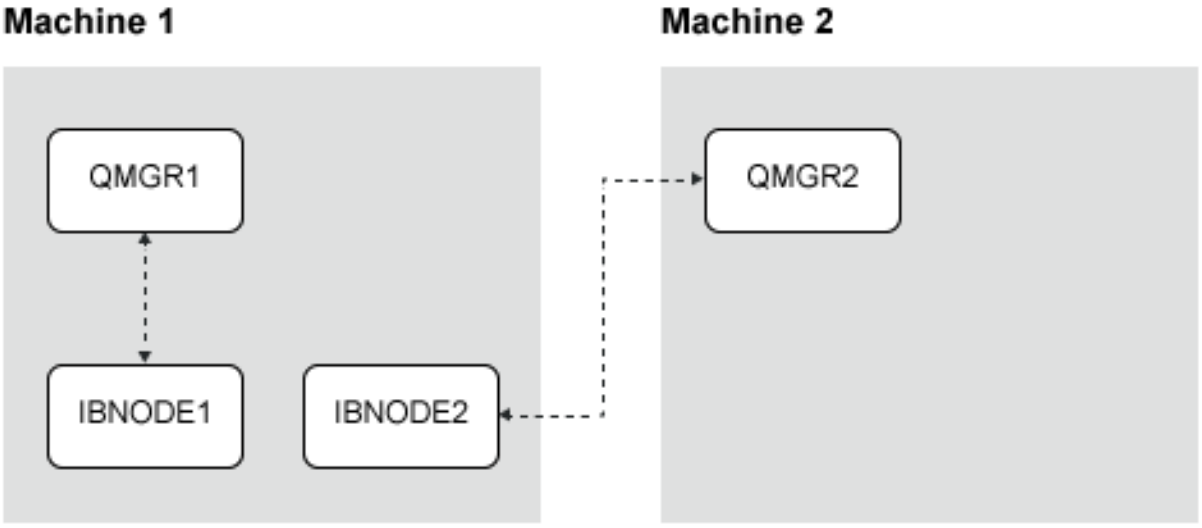
In this diagram, integration node IBNODE1 and IBNODE2 have client connections to the QMGR1 queue manager. The integration nodes are configured as a failover as part of a high availability IBM MQ solution, handling the messaging processing and saved states for QMGR1. To learn more about high availability configurations, see "Configuring integration nodes for high availability" on page 120.

Table 14. Strengths and weaknesses of scenario 2:

Strengths	Weaknesses
• The integration node can connect to a queue manager, both local and remote, without changing an existing IBM MQ topology • High availability IBM MQ-based solution	• Failover: there is a single point of failure (QMGR1). To mitigate this risk, you can add more queue managers

Scenario 3

This scenario uses a single integration node to a single queue manager.



In this diagram, the relationship between integration node and queue managers is 1:1. A 1:1 locally connected configuration was the only option that was available before the release of IBM Integration Bus 10.0.

Table 15. Strengths and weaknesses of scenario 3:	
Strengths	**Weaknesses**
• High level of control of connection behavior between IBM App Connect Enterprise and IBM MQ	• Inflexible topology, requiring more manual effort to control connection behavior and maintain solution

For more information about queue manager architecture, see Designing an IBM MQ architecture in the IBM MQ product documentation online

Configuring connections to IBM MQ

You can configure a local or client connection to IBM MQ to enable your IBM App Connect Enterprise message flows to access messages on IBM MQ queues.

If you intend to use an integration node (or an integration server that is managed by an integration node) to process IBM MQ messages, you must install IBM MQ in addition to installing IBM App Connect Enterprise. If you want to connect to a remote queue manager to process IBM MQ messages, you must install either an MQ Client or an MQ Server on the same machine as IBM App Connect Enterprise, in addition to installing an MQ Server on the machine that is running your queue manager. IBM MQ is available as a separate installation package, and your IBM App Connect Enterprise license entitles you to install and use IBM MQ with IBM App Connect Enterprise. For more information, see “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “Installing IBM MQ” on page 96.

Alternatively, if you are using an independent integration server, you can configure a remote default queue manager to process the IBM MQ messages, by specifying the name of an MQEndpoint policy in the `server.conf.yaml` configuration file for the integration server. For more information, see “Configuring an integration server to use a remote default queue manager” on page 751.

You can configure MQ Connection properties on the following nodes to specify a local or client connection with IBM MQ:

- node
- node
- node
- node

You can also specify the properties through an policy.

You can connect to a secured IBM MQ queue manager (local or remote), by passing a user name and password to the queue manager when the connection is made. You can also choose whether to use the SSL protocol when a client connection is made to a remote queue manager. For information about securing connections to IBM MQ, see “Connecting to a secured IBM MQ queue manager” on page 747.

You can choose to configure either a local or client connection between your integration node and your queue manager, depending on the configuration of your existing architecture. If your queue manager is running on the same machine as your integration node, you can specify a local connection, either by choosing a specific queue manager to be used for the MQ node or by using the queue manager that is specified on the integration node. Alternatively, if IBM MQ is installed on separate machines from IBM App Connect Enterprise, you can define a client connection, either by configuring the connection details on the MQ node or policy, or by specifying a client channel definition table (CCDT) to control the client connection information. For more information, see “Configuring a client connection to IBM MQ” on page 745.

You can specify either a local or client connection by using one of the following methods:

- Set explicit connection properties for a specific MQ node on the MQ Connection tab.
- Specify an MQEndpoint policy to control connection properties for one or more MQ nodes on the Policy tab.

An MQEndpoint policy dynamically controls the connection properties that are applied at run time. You can use a single MQEndpoint policy for multiple MQ nodes, so that when you update the connection properties in the MQEndpoint policy, the updated properties are automatically applied to all MQ nodes that have the policy attached.

You can also configure an independent integration server to use a remote default queue manager for interactions with IBM MQ, by specifying the name of an MQEndpoint policy in the **remoteDefaultQueueManager** property in the integration server's `server.conf.yaml` file. This property enables the connection details that are set in the MQEndpoint policy to be used by default when connections to IBM MQ are required by flows that are deployed to the integration server. For more information, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

All MQ nodes that do not have either MQ Connection properties set, or an MQEndpoint policy specified, use the connection details of the queue manager that is associated with the integration node or server at run time. If no queue manager was specified for the integration node, the message flow cannot deploy.

If you want to set explicit connection properties, select a value for the `Connection` property on the MQ Connection tab:

- Select `Local queue manager` to specify a local connection to a named queue manager, which is specified in the `Destination queue manager name` property.
- Select `MQ client connection properties` to choose a client connection to a queue manager, and specify the connection details of the queue manager in the following properties:
  - `Queue manager host name`
  - `Listener port number`
  - `Channel name`
  - `Destination queue manager name`
- Select `Connect with CCDT` to use the client connection details that are specified in a client channel definition table (CCDT).

The following topics describe how to configure local and client connections to IBM MQ:

- "Configuring a local connection to IBM MQ" on page 744
- "Configuring a client connection to IBM MQ" on page 745
- "Connecting to a secured IBM MQ queue manager" on page 747
- "Configuring an integration server to use a remote default queue manager" on page 751

By default, a thread that is processing IBM MQ messages becomes idle when it has not received any messages on its input queue for 1 minute, at which point the connection times out. However, you can change the length of time after which a connection for an idle message flow is released, by setting the **sharedConnectorIdleTimeout** property of the **mqsichangeproperties** command. For more information, see connections.

Configuring a local connection to IBM MQ
You can configure a local connection to IBM MQ by specifying the Connection properties on the MQInput, MQOutput, MQGet, or MQReply node.

Before you begin

Complete the following tasks:

- Read the topic "Configuring connections to IBM MQ" on page 743.
- Ensure that the required queue manager has been created, and that the user ID that is running the integration node has the necessary permissions to access the queue manager.
- Decide how you want to connect to the queue manager. You can either configure the connection properties on the MQ Connection tab of the MQ node, or you can use an MQEndpoint policy on the Policy tab. The steps in this topic explain how to configure connections by using the properties on the MQ

Connection tab. The properties that you can set in a policy are listed in the MQ node topics, such as the node; for more information about using an MQEndpoint policy, see policy.

About this task

IBM MQ is available as a separate installation package, and your IBM App Connect Enterprise license entitles you to install and use IBM MQ with IBM App Connect Enterprise. For more information, see “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “Installing IBM MQ” on page 96.

If your IBM MQ queue manager is running on the same machine as IBM App Connect Enterprise, you can specify a local connection to enable your IBM App Connect Enterprise message flows to access messages on IBM MQ queues.

Procedure

1. On the MQ Connection tab, select the `Local queue manager` property to specify a local connection to a named local queue manager.

   If you do not set any MQ Connection properties, the queue manager that is specified on the integration node is used by default. If the MQ Connection properties are not set and no queue manager is specified on the integration node, the flow fails to deploy.
2. Specify the name of the required queue manager in the `Destination queue manager name` property.
3. Configure the connection to use a security identity for authentication. For more information, see “Connecting to a secured IBM MQ queue manager” on page 747.

What to do next

For MQInput nodes, the connection is made when the flow is deployed. For MQOutput, MQGet, and MQReply nodes, the connection is made when the first message is sent or received.

If you later decide that you want to control connection properties by using an MQEndpoint policy, property values that are set on the MQ Connection tab are ignored when a policy is attached to the message flow node. If you want to revert to using the connection details of the queue manager that was specified on the integration node, remove the settings for the **Policy URL** and MQ Connection tab properties.

Configuring a client connection to IBM MQ

On distributed systems, you can configure a client connection to IBM MQ by defining a connection channel and listener on IBM MQ and setting the MQ Connection properties of the MQ node on IBM App Connect Enterprise.

Before you begin

Complete the following tasks:

- Read the topic “Configuring connections to IBM MQ” on page 743.
- Ensure that the required queue manager has been created on the IBM MQ server.
- Ensure that the user ID that is running the integration node has the necessary permissions to access the queue manager.
- Decide how you want to connect to the queue manager. You can either configure the connection properties on the MQ Connection tab of the MQ node (such as the node), or you can specify an MQEndpoint policy on the Policy tab. The properties set in the MQEndpoint policy override the connection properties on the node at run time. The steps in this topic explain how to configure connections by using the properties on the MQ Connection tab. For information about the properties that you can set in a policy, see policy.

  For independent integration servers, you can configure a remote default queue manager for interactions with IBM MQ, by specifying the name of an MQEndpoint policy in the **`remoteDefaultQueueManager`** property in the integration server's `server.conf.yaml` file. This property enables the connection details that are specified in the MQEndpoint policy to be used by default when MQ connections are

required by flows that are deployed to the integration server. For more information, see “Using a remote default queue manager” on page 750 and “Configuring an integration server to use a remote default queue manager” on page 751.

About this task

IBM MQ is available as a separate installation package, and your IBM App Connect Enterprise license entitles you to install and use IBM MQ with IBM App Connect Enterprise. For more information, see “Enhanced flexibility in interactions with IBM MQ ” on page 26 and “Installing IBM MQ” on page 96.

If IBM MQ is running on a separate machine from IBM App Connect Enterprise, you can configure a client connection so that your integration node or server can access messages on the remote queue managers.

You can define a client connection on the MQ node either by specifying the connection details of the queue manager explicitly, through the `MQ client connection properties`, or by selecting the `Connect with CCDT` property to use a client channel definition table (CCDT) that contains the details of the connection. The MQ Connection properties are available on the following MQ nodes:

- MQInput
- MQOutput
- MQGet
- MQReply

Procedure

You configure a client connection by configuring the connection channel and listener on IBM MQ, and setting the MQ Connection properties on IBM App Connect Enterprise.

On the machine that is running your IBM MQ queue manager:

1. Define a server-connection channel for the queue manager, and ensure that the number of shared conversations is set to a minimum of 10.

   The number of conversations that can share a TCP/IP connection is specified by the **SHARECNV** property on the IBM MQ channel, and the default for this property is 10. If too few shared conversations are defined on the channel, errors can occur when the MQ node attempts to connect to the queue. For information about how to define a channel, see the IBM MQ product documentation online.
2. Define a TCP/IP listener.

   For information about how to define a listener, see the IBM MQ product documentation online.

Define the client connections on the machine that is running your integration server:

3. Decide how you want to define the connections to the queue manager.

   You can either configure them on the **MQ Connection** tab of the MQ node, or you can control them by using an MQEndpoint policy, specified on the **Policy** tab. The remaining steps in this topic explain how to configure connections by using the properties on the **MQ Connection** tab. For information about the MQ connection properties that you can set in a policy, see policy.
4. On the **MQ Connection** tab, choose one of the following options for the `Connection` property:

   - Select `MQ client connection properties` to make a client connection to the queue manager by explicitly specifying the connection details of the target queue manager. Specify the following connection properties for the target queue manager:

     – `Queue manager host name`

     – `Listener port number`

     – `Channel name`

     – `Destination queue manager name`

5. Configure the connection to use a security identity for authentication, SSL for confidentiality, or both, by completing the steps described in "Connecting to a secured IBM MQ queue manager" on page 747.

What to do next

All IBM MQ connections remain open until one of the following events occurs:

- An error occurs while the queue manager is being accessed, and a new connection is required as a result.
- The message flow is stopped.
- The message flow is idle for a time that exceeds the timeout threshold set for the connection. By default, this threshold is set to 1 minute; however, you can change the threshold by setting the **sharedConnectorIdleTimeout** property of the **mqsichangeproperties** command. For more information, see connections.

If you later decide that you want to control connection properties by using an MQEndpoint policy, property values that are set on the MQ Connection tab are ignored when a policy is attached to the message flow node.

Connecting to a secured IBM MQ queue manager
You can configure a connection to a secured local or remote IBM MQ queue manager, by setting properties on an MQ node or in an MQEndpoint policy.

Before you begin

- Read the topic "Configuring connections to IBM MQ" on page 743.
- Ensure that the required queue manager has been created on the IBM MQ server.
- Ensure that the user ID that is running the integration node has the necessary permissions to access the queue manager.

About this task

When you configure an MQ connection from an MQ node to an IBM MQ queue manager, you can optionally configure the connection to use a security identity for authentication, SSL for confidentiality, or both. The security identity, which passes user name and password security credentials to the queue manager, can be used on connections to local or remote queue managers. For connections to remote queue managers, you can choose whether to use the SSL protocol to provide confidentiality on the client connection. IBM App Connect Enterprise supports a subset of the SSL functionality that is supported by IBM MQ.

You can use the `Security identity` property on the MQ node or policy to pass a user name and password to the queue manager, by specifying a security identity that contains those credentials. The identity is defined using the **mqsisetdbparms** command.

You can specify that the SSL protocol is to be used when a client connection is made to a remote queue manager, by selecting the `Use SSL` property on the MQ node or policy. You can use SSL for client connections that are configured using either the `MQ client connection properties` or a client channel definition table (CCDT). If you specify SSL on the client connection, you must also specify the location of the SSL key repository. The SSL key repository is created by using the IBM MQ GSKit, and it holds the required private and public certificates appropriate to the chosen certificate policy for the queue manager. The SSL key repository password stash file *key repository file name*`.sth`, which is created using IBM MQ GSKit, must be located in same folder as the key repository.

You can use the **SSL certificate label** property in the MQEndpoint policy to specify the label of the certificate to be used when establishing the SSL client connection to the queue manager. If no value is supplied, the certificate with the MQ default label of `ibmwebspheremq`*user_id* is used, where *user_id* is the user identifier of the integration server. When multiple MQ SSL certificates are used by the same integration server, the MQ SVRCONN channel must be configured with a SHARECNV value of either `0` or `1`, to ensure that the correct certificate is used for the channel.

You can define the security properties for a local or client connection on an MQ node by using the MQ Connection properties on the following nodes:

- MQInput
- MQOutput
- MQGet
- MQReply

You can also set the security properties for an MQ connection by using an MQEndpoint policy, and these values override the values of properties set on the node, at run time. For more information, see policy.

Procedure

Follow these steps to complete the configuration of the integration server:

1. If your IBM MQ queue managers require a user name and password, you can use the **mqsisetdbparms** command to provide them for the secured connection. You can specify these credentials for all MQ connections (`mq::MQ`), for all connections to a specified queue manager (`mq::QMGR::`*`QMName`*), or for a connection with a specified security identity (`mq::`*`securityIdentityName`*). If you intend to use a security identity to provide user name and password information for a connection to a secured queue manager, you can use the **mqsisetdbparms** command to define that identity. The name of this identity can then be referred to by the **Security identity** property in the MQ nodes or MQEndpoint policy, as a method of retrieving credentials for a secured connection. When you set the security identity by using this command, ensure that it is prefixed by `mq::`.
   For example:
   - Create a security identity to be used for retrieving user name and password credentials when making a connection:

     ```
     mqsisetdbparms -w workDir -n mq::securityIdentityName -u username -p password
     ```

     For example, if you use this command to create an identity called `myNodeMQCreds`, you can configure the MQ node to use the credentials associated with this identity by specifying the name `myNodeMQCreds` in the **Security identity** property of the MQ node or policy.
   - Configure a user name and password to be used for all MQ connections to a named queue manager (local or client connections), when no security identity name has been specified in the MQ node or policy:

     ```
     mqsisetdbparms -w workDir -n mq::QMGR::QMName  -u username -p password
     ```

     For example, if you know that all connections to a queue manager called `mySecureQM` will require a user name and password, you can specify that all connections to that queue manager will use the user name and password specified by the **mqsisetdbparms** command:

     ```
     mqsisetdbparms -w workDir -n mq::QMGR::mySecureQM  -u myUsername -p myPassword
     ```
   - Configure a user name and password for all MQ connections (local or client connections) where no security identity name has been set on the MQ node or policy, and where the queue manager that

is being connected to does not match any queue manager names that have been specified using `mq::QMGR::QMName`:

```
mqsisetdbparms -w workDir -n mq::MQ  -u username -p password
```

If no security identity has been specified, no credentials have been set for the queue manager (`mq::QMGR`), and no default credentials set for MQ (`mq::MQ`), no user name and password are passed to the queue manager, and the connection to the secured queue manager fails as a result.

Do not include the `mq::` prefix when setting the security identity on the MQ node or in the policy.

You can use the **mqsireportdbparms** to find out which security credentials have been set for the MQ connection. For example:

```
mqsireportdbparms -w workDir -n mq::*
```

2. If you are using SSL for any MQ connections, specify the location of the key repository by using the **mqsichangeproperties** command.

   This value is specified as the full file path of the SSL key repository minus the `.kdb` file extension. For example, if the SSL key repository is `C:\SSL\key.kdb`, set the location of the key repository by using the following command:

   ```
   mqsichangeproperties ACE11NODE -o BrokerRegistry -n mqKeyRepository -v C:\SSL\key
   ```

   a) Ensure that the SSL key repository password stash file *key repository file name*`.sth` is located in same folder as the key repository.

   This stash file is created using IBM MQ GSKit.

   b) Use the MQSC `REFRESH SECURITY` command to enable the changes to the SSL key repository to take effect.

Follow these steps to complete the required connection configuration in the MQ node or policy:

3. Configure either a local or client connection to the queue manager, as described in one of the following topics:

   - "Configuring a local connection to IBM MQ" on page 744
   - "Configuring a client connection to IBM MQ" on page 745

4. Use the `Security identity` property to provide the user name and password on a specific connection to the secured queue manager, through the security identity that you created by using the **mqsisetdbparms** command.

   The value that you set in this property is the name of the security identity that you defined by using the **mqsisetdbparms** command in step 1.

   If you do not specify the `Security identity` property, the security credentials that have been set for all MQ connections (`mq::MQ`) or for all connections to a specified queue manager (`mq::QMGR::QMName`), will be used, if appropriate.

   You can use the `Security identity` property to provide the security credentials on local and client connections. This property is not available for client connections that use a client channel definition table (CCDT); for these connections, specify the required information in the CCDT.

5. If you are configuring a client connection to a remote queue manager, you can choose whether to use the SSL protocol when a client connection is made to a remote queue manager.
   a) Select the `Use SSL` property on the MQ node to provide confidentiality on the client connection, by using SSL.

      This property is available for client connections that are configured using either the `MQ client connection properties` or a client channel definition table (CCDT).
   b) Specify the `SSL peer name` property, which specifies the name that is passed to the remote queue manager when making the client connection. There must be a positive match for the connection to succeed.

      This property is available only if the client connection details are specified through the `MQ client connection properties`; if the client connection uses a client channel definition table (CCDT), you can specify this information in the CCDT.
   c) Specify the `SSL cipher specification` property, which specifies the name of the symmetric key cryptography algorithm through which the remote queue manager is secured.

      This property is available only if the client connection details are specified through the `MQ client connection properties`; if the client connection uses a client channel definition table (CCDT), you can specify this information in the CCDT.

What to do next

The MQInput node attempts to connect to the queue manager when the flow is deployed and started. The MQOutput, MQGet, and MQReply nodes attempt to connect when the first message is sent or received. If any connection problems occur, see the IBM MQ product documentation for information about any `mqrc` return code values that are reported in the IBM App Connect Enterprise BIP messages.

If you later decide that you want to control connection properties by using an MQEndpoint policy, you can attach a policy to the message flow node. Property values that are set on the MQ Connection tab are ignored when a policy is attached to the message flow node.

Using a remote default queue manager
You can configure an independent integration server to connect to a default queue manager on a remote machine that is running IBM MQ.

Several IBM App Connect Enterprise capabilities use IBM MQ, such as the MQ nodes (MQInput, MQOutput, and MQGet), statistics and monitoring events, and internal state storage for such things as event-driven processing nodes (aggregation, sequencing, collection, and timer nodes). In App Connect Enterprise, interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting.

By using a remote default queue manager, you can configure independent integration servers to direct all MQ traffic to a remote MQ server, without configuring policy settings on each message flow node. You can also configure independent integration servers to share queue managers, which simplifies IBM MQ administration. The MQ topology does not need to be adjusted to allow a local queue manager for App Connect Enterprise integration servers, and App Connect Enterprise integration servers can be provisioned independently of the MQ queue managers.

As the default queue manager can be used to store state for the integration server itself, App Connect Enterprise shuts down if the queue manager becomes unavailable; this applies to remote default queue managers and local default queue managers. In a container-based environment with remote queue managers provisioned separately, the container management infrastructure is responsible for restarting the App Connect Enterprise integration server if the queue manager becomes unavailable, and for providing the new instance of the integration server with a working queue manager connection. The integration server will not start if the remote default queue manager is unavailable.

The remote default queue manager is used only if no local default queue manager has been configured for the integration server. If a message flow interacts directly with MQ, the remote default queue manager is used only if there is no explicit MQ configuration and no policy associated with the message flow nodes.

Properties that are set on the message flow node take precedence over properties set in a remote default queue manager policy.

For statistics and monitoring functions of the integration server, explicit configuration takes precedence over the default queue manager settings. For internal state storage, the remote default queue manager is used if there is no local default queue manager, and you can use the replacement queue prefix setting to maintain separate state stores for different servers using the same queue manager. The internal queues are typically created in advance for the local default queue manager (using sample scripts provided with the product), but they are created automatically if permissions allow this for both local and remote default queue managers. You can also create queues in advance with a different prefix by modifying the sample scripts and running them on a shared queue manager. From an MQ queue manager perspective, the App Connect Enterprise integration server appears as a standard client application, and all the standard client configuration options are supported. As with other MQ client applications, the queue manager does not provide XA transactional coordination. However, security and performance tuning parameters are supported, as described in the IBM MQ product documentation online.

Only independent integration servers can be configured to use a remote default queue manager; for interactions between integration nodes (or integration servers that are managed by an integration node) and IBM MQ, you must install IBM MQ on the same machine as the integration node, and use a local default queue manager to access the MQ queues.

For information about how to configure a connection to a remote default queue manager, see "Configuring an integration server to use a remote default queue manager" on page 751.

For more information about using IBM App Connect Enterprise with IBM MQ, see the following topics:

- "Processing IBM MQ messages" on page 738
- "IBM MQ topologies" on page 739
- "Configuring connections to IBM MQ" on page 743
- "Configuring a client connection to IBM MQ" on page 745
- "Connecting to a secured IBM MQ queue manager" on page 747

Configuring an integration server to use a remote default queue manager
You can configure an independent integration server to connect to a default queue manager on a remote machine that is running IBM MQ.

Before you begin

Read the following topics:

- "Processing IBM MQ messages" on page 738
- "IBM MQ topologies" on page 739
- "Configuring connections to IBM MQ" on page 743

About this task

In IBM App Connect Enterprise, interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting.

You can configure an independent integration server to use a remote default queue manager by setting the **`remoteDefaultQueueManager`** property in the `server.conf.yaml` file. This property enables an MQEndpoint policy to be used to specify an MQ client connection, which is then used by default when a connection to IBM MQ is made. You can use the **`replacementQueuePrefix`** property in the `server.conf.yaml` file to configure the integration server to use the specified prefix instead of the usual `SYSTEM.BROKER.` queue prefix for MQ queues.

For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750. For more information about IBM MQ, see the IBM MQ product documentation online.

Procedure

Configure an independent integration server to use a remote default queue manager, by completing the following steps:

1. Create an MQEndpoint policy, as described in "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327. Set the **Connection** property to `Client`, and specify values for the **Queue manager name**, **Queue manager host name**, **Listener port number**, and **Channel name** properties.

   For more information about the properties that you can set in an MQEndpoint policy, see policy.
2. Deploy the policy project to the work directory for the integration server, either by using the command to deploy the BAR file that contains the policy project, or by copying the `.policyxml` file into the run sub-directory of the integration server work directory, in a sub-directory that has the name of the policy project.
3. Set up any security credentials that are required for the remote queue manager connection, as described in "Connecting to a secured IBM MQ queue manager" on page 747.
4. Configure the integration server to use the remote default queue manager, by modifying the integration server's `server.conf.yaml` file:
   a) Open the `server.conf.yaml` configuration file for your integration server, by using a YAML editor.

      If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

      The `server.conf.yaml` file for an independent integration server is located in the root of the integration server's work directory (`<work directory>/server.conf.yaml`).
   b) Specify the remote default queue manager to be used for the integration server, by setting the **remoteDefaultQueueManager** property to the policy name and project of the MQEndpoint policy that you created in step 1, in the format `{policyProjectName}:policyName`; for example:

      ```
      remoteDefaultQueueManager: '{myPolicyProject}:policy1'
      ```
   c) Optional: If the queue manager is shared by multiple integration servers, you can use the **replacementQueuePrefix** property to specify a unique queue prefix for each integration server, to prevent them from sharing the same queues. Specify this property in the format `replacementQueuePrefix: 'MYPREFIX1'`. This prefix then replaces the `SYSTEM.BROKER.` section of each queue name.

      If the required App Connect Enterprise queues have not already been created on the remote queue manager, and if security restrictions prevent the integration server from creating those queues, you can create them by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99. If you are using non-default queue names (by specifying a replacement queue prefix), you must modify the queue names in the sample MQSC script `iib_queues_create.mqsc`, which is used by the **iib_createqueues** command to define the queues.
   d) Save the modified `server.conf.yaml` file.

   For more information about configuring an integration server by setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
5. Restart the integration server for the changes to take effect.

Working with IBM MQ

You can use the IBM App Connect Enterprise MQ nodes to handle messages that are received from or sent to IBM MQ applications, enabling you to perform various tasks.

About this task

IBM MQ is available as a separate installation package, and your IBM App Connect Enterprise license entitles you to install and use IBM MQ with IBM App Connect Enterprise. For more information, see "Installing IBM MQ" on page 96.

The following topics provide step-by-step information about how to complete a set of common tasks that involve IBM MQ messages:

- "Receive an MQ datagram message, transform the message, and pass it to an MQ queue" on page 753
- "Receive an MQ request message, pass the message to MQ, and then pass the response from MQ to the requesting client" on page 755

Receive an MQ datagram message, transform the message, and pass it to an MQ queue

Transform IBM MQ messages in IBM App Connect Enterprise in a one-way message pattern.

Before you begin

For this scenario, server bindings are used on the MQ nodes to connect to IBM MQ. IBM MQ therefore must be installed on the same computer as the integration node. Alternatively, you can configure the MQ nodes for remote connections, but these steps are not described within the scenario. For more information, see "Configuring connections to IBM MQ" on page 743.

About this task

This scenario is used to create a one-way messaging pattern.

Scenario:

In an existing IBM MQ messaging topology, you can use IBM App Connect Enterprise to transform an IBM MQ message, and pass it back to a different queue. No reply confirmation is sent; the message is processed and immediately passed to the specified queue. If a reply is required, follow the steps in "Receive an MQ request message, pass the message to MQ, and then pass the response from MQ to the requesting client" on page 755.

For this scenario, there is no message content checking or error handling configured for the message flow nodes, because suitable actions depend on the overall messaging system architecture.

Instructions:

The following steps show how to create a message flow so that a message from IBM MQ can be transformed, and then put to another queue.

Procedure

1. Create a message flow that is named MQ_Task1_One_Way with the following nodes:
   - An MQInput node.
   - A Compute node.
   - An MQOutput node.

   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the Compute node.

3. Connect the Out terminal of the Compute node to the In terminal of the MQOutput node.

You must now configure the message flow nodes.

4. Set the following properties of the MQInput node:
   a) On the **Basic** tab, set the `Queue name` property to `MQ.TASK1.IN1`.
   b) On the **MQ Connection** tab, set the `Connection` property to `Local queue manager`, and then the `Destination queue manager` property to `TASK1_QUEUE_MANAGER`.
   c) On the **Input Message Parsing** tab:
      - Set the `Message domain` property to `XMLNSC`.
5. Set the following properties of the Compute node:
   a) An ESQL module, `MQ_Task1_One_Way_Compute`, is automatically created. In the Application Development view, open **ESQLs**, and then the `MQ_Task1_One_Way_Compute` module.
   b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task1_One_Way_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  SET OutputRoot.XMLNSC.Order.TimeStamp = CURRENT_GMTTIMESTAMP;
  RETURN TRUE;
 END;
END MODULE;
```

   The Compute node represents the solution transformation in this scenario.

   The Compute node adds a time stamp value to the message, which shows the time that it was processed.
6. Set the following properties of the MQOutput node:
   a) On the **Basic** tab, set the `Queue name` property to `MQ.TASK1.OUT1`.
   b) On the **MQ Connection** tab, set the `Connection` property to `Local queue manager`, and then the `Destination queue manager` property to `TASK1_QUEUE_MANAGER`.
7. Save the message flow.

You can now test the message flow. Create an IBM MQ message, send the message to the input queue, and observe the changes to the message on the output queue.

8. Create an IBM MQ message with a Payload that contains XML for the input queue in the following format:

```
<SaleEnvelope>
 <OrderNum>123</OrderNum>
 <CustomerNum>456</CustomerNum>
 <ProductCode>789</ProductCode>
 <Qty>20</Qty>
</SaleEnvelope>
```

9. Run the message flow.

Results

When the message flow is run, the following actions occur:

1. The MQInput reads an IBM MQ message in from an application.
2. The Compute node transforms the message.

3. The MQOutput node puts the transformed message onto the specified output queue.

Receive an MQ request message, pass the message to MQ, and then pass the response from MQ to the requesting client
Receive an IBM MQ request message, pass the message to IBM MQ, and then send the IBM MQ response to the requesting client.

Before you begin

For this scenario, server bindings are used on the MQ nodes to connect to IBM MQ. IBM MQ therefore must be installed on the same computer as the integration node. Alternatively, you can configure the MQ nodes for remote connections, but these steps are not described within the scenario. For more information, see "Configuring connections to IBM MQ" on page 743.

About this task

A request-response message flow is a specialized form of a point-to-point application. For a general description of these applications, see "Nodes for connectivity" on page 487.

This scenario is used for connecting client applications.

Scenario:

Two applications with different message formats communicate with each other by using IBM MQ in a request-reply message processing solution. For the applications to communicate successfully, the message formats must be transformed in IBM App Connect Enterprise on both the request and reply messages. The backend application is represented in this scenario by a third message flow, to demonstrate the successful processing of a message.

Two message flows are required for request-reply, because IBM MQ processes messages asynchronously. Messages that are processed asynchronously in IBM App Connect Enterprise do not wait for a response, and IBM MQ does not send a reply message. To process a reply, a store queue takes the original message's `ReplyToQ` property so that a second message flow can process a reply from an application.

This scenario demonstrates how to create Request and Reply message flows, and how to store message state information by using a store queue. The message flow that represents the backend application modifies a "Completion Time" value in the message.

Instructions:

The following steps show how to create the message flows and a store state queue so that a reply message from IBM MQ can be processed.

Procedure

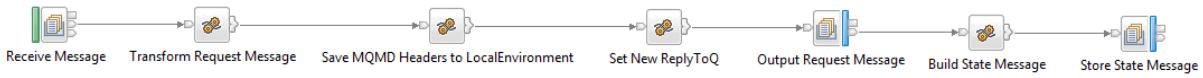
1. Create a message flow that is named MQ_Task2_Request with the following nodes, and name them on the **Description** tab `Node name` property:
   - An MQInput node, named `Receive Message`.
   - Four Compute nodes, named `Transform Request Message`, `Save MQMD Headers to LocalEnvironment`, `Set New ReplyToQ`, and `Build State Message`.
   - Two MQOutput nodes, named `Output Request Message` and `Store State Message`.

   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the `Receive Message` MQInput node to the In terminal of the `Transform Request Message` Compute node.
3. Connect the Out terminal of the `Transform Request Message` Compute node to the In terminal of the `Save MQMD Headers to LocalEnvironment` Compute node.

   Note: You can add your own transformation node, or a subflow, for the message processing that you require between the `Receive Message` MQInput node and `Save MQMD Headers to`

LocalEnvironment Compute node. The Transform Request Message Compute node is a placeholder in this scenario that does not transform the message.

4. Connect the Out terminal of the Save MQMD Headers to LocalEnvironment Compute node to the In terminal of the Set New ReplyToQ Compute node.
5. Connect the Out terminal of the Set New ReplyToQ Compute node to the In terminal of the Output Request Message MQOutput node.
6. Connect the Out terminal of the Output Request Message MQOutput node to the In terminal of the Build State Message Compute node.
7. Connect the Out terminal of the Build State Message Compute node to the In terminal of the Store State Message MQOutput node.



You must now configure the message flow nodes for the Request behavior of the message flow.

8. Set the following properties of the Receive Message MQInput node:
   a) On the **Basic** tab, set the Queue name property to MQ.TASK2.IN1.
   b) On the **MQ Connection** tab, set the Connection property to Local queue manager, and then the Destination queue manager property to TASK2_QUEUE_MANAGER.
   c) On the **Input Message Parsing** tab, set the Message domain property to XMLNSC.
9. Set the following properties of the Transform Request Message Compute node:
   a) An ESQL module, MQ_Task2_Request_Transform_Reply_Message, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_Request_Transform_Reply_Message module.
   b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Request_Transform_Reply_Message
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  RETURN TRUE;
 END;
END MODULE;
```

   The Transform Request Message Compute does not transform the message, and acts as a placeholder for this scenario.
10. Set the following properties of the Save MQMD Headers to LocalEnvironment Compute node:
    a) On the **Basic** tab, set the Compute mode property to LocalEnvironment.
    b) An ESQL module, MQ_Task2_Request_Save_MQMD_Headers_to_LocalEnvironment, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_Request_Save_MQMD_Headers_to_LocalEnvironment module.
    c) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Request_Save_MQMD_Headers_to_LocalEnvironment
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputLocalEnvironment.Variables.OriginalMQMD = InputRoot.MQMD;
  RETURN TRUE;
 END;
END MODULE;
```

    The Save MQMD Headers to LocalEnvironment Compute node saves the MQMD of the original message to a local environment variable, which is then retrieved later in the message flow by the Build State Message Compute node.

11. Set the following properties of the Set New ReplyToQ Compute node:
    a) An ESQL module, MQ_Task2_Request_Set_New_ReplyToQ, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_Request_Set_New_ReplyToQ module.
    b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Request_Set_New_ReplyToQ
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  SET OutputRoot.MQMD.ReplyToQ = 'MQ.TASK2.BACKEND.RESP1';
  SET OutputRoot.MQMD.ReplyToQMgr = 'TASK2_QUEUE_MANAGER';
  RETURN TRUE;
 END;
END MODULE;
```

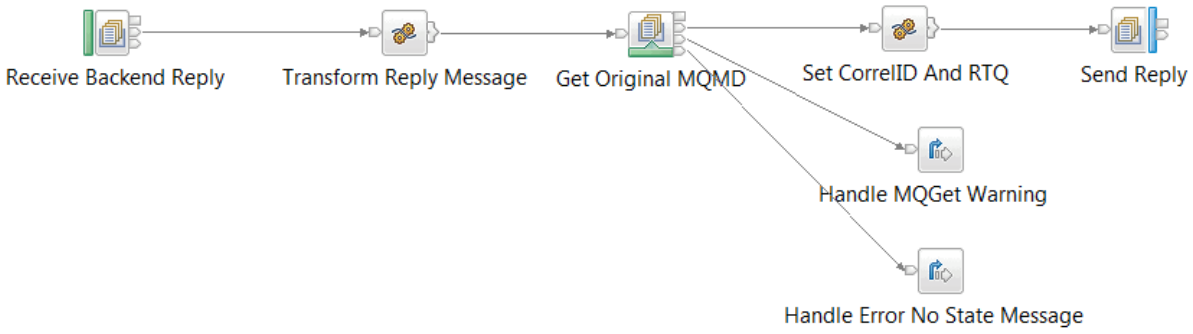
    The Set New ReplyToQ Compute node sets a new value for ReplytoQ so that the Reply message flow queue is used.
12. Set the following properties of the Output Request Message MQOutput node:
    a) On the **Basic** tab, set the Queue name property to MQ.TASK2.BACKEND.REQ1.
    b) On the **Advanced** tab, select the New message ID property.
13. Set the following properties of the Build State Message Compute node:
    a) An ESQL module, MQ_Task2_Request_Build_State_Message, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_Request_Build_State_Message module.
    b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Request_Build_State_Message
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();
  SET OutputRoot.MQMD.CorrelId =
 InputLocalEnvironment.WrittenDestination.MQ.DestinationData.msgId;
  SET OutputRoot.MQMD.ReplyToQ = InputLocalEnvironment.Variables.OriginalMQMD.ReplyToQ;
  SET OutputRoot.MQMD.ReplyToQMgr =
 InputLocalEnvironment.Variables.OriginalMQMD.ReplyToQMgr;
  SET OutputRoot.BLOB.BLOB = InputLocalEnvironment.Variables.OriginalMQMD.MsgId;
  RETURN TRUE;
 END;

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

    The Build State Message Compute node:
    i) Extracts the Message ID of the message that is written to MQ.TASK2.BACKEND.REQ1.
    ii) Extracts the original ReplyToQ and ReplyToQMgr values from the LocalEnvironment, and these values are saved in the state message MQMD.
    iii) Extracts the original MessageId value from the LocalEnvironment, and saves this value to the body of the state message.
    iv) Uses the BLOB parser to read and write the body of the state message, because the MessageId is binary data.
14. Set the following properties of the Store State Queue MQOutput node:
    a) On the **Basic** tab, set the Queue name property to MQ.TASK2.STATE1.
    b) On the **Advanced** tab, select the New message ID property.

You must now create the Reply message flow.

15. Create a message flow that is named MQ_Task2_Reply with the following nodes, and name them on the **Description** tab `Node name` property:
    - An MQInput node, named `Receive Backend Application Reply`.
    - An MQGet node, named `Get Original MQMD`.
    - An MQOutput node, named `Send Reply`.
    - Two Compute nodes, named `Transform Reply Message` and `Set CorrelID And RTQ`.
    - Two Throw nodes, named `Handle MQGet Warning` and `Handle Error No Store Message`.
16. Connect the Out terminal of the `Receive Backend Application Reply` MQInput node to the In terminal of the `Transform Reply Message` Compute node.
17. Connect the Out terminal of the `Transform Reply Message` Compute node to the In terminal of the `Get Original MQMD` MQGet node.
18. Connect the Warning terminal of the `Get Original MQMD` MQGet node to the `Handle MQGet Warning` Throw node.
19. Connect the Failure terminal of the `Get Original MQMD` MQGet node to the `Handle Error Not Stored Message` Throw node.
20. Connect the Out terminal of the `Get Original MQMD` MQGet node to the `Set CorrelID And RTQ` Compute node.
21. Connect the Out terminal of the `Set CorrelID And RTQ` Compute node to the `Send Reply` MQOutput node.



You must now configure the message flow nodes for the Reply behavior of the message flow.

22. Set the following properties of the `Receive Backend Application Reply` MQInput node:
    a) On the **Basic** tab, set the `Queue name` property to `MQ.TASK2.BACKEND.RESP1`.
    b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
23. Set the following properties of the `Transform Reply Message` Compute node:
    a) An ESQL module, `MQ_Task2_Reply_Transform_Reply_Message`, is automatically created. In the Application Development view, open **ESQLs**, and then the `MQ_Task2_Reply_Transform_Reply_Message` module.
    b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Reply_Transform_Reply_Message
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  RETURN TRUE;
 END;
END MODULE;
```

    The Compute does not transform the message. The Compute node is a placeholder for the solution transformation in this scenario.

24. Set the following properties of the Get Original MQMD MQGet node:
    a) On the **Basic** tab, set the Queue name property to MQ.TASK2.STATE1.
    b) On the **MQ Connection** tab, set the Connection property to Local queue manager, and then the Destination queue manager property to TASK2_QUEUE_MANAGER.
    c) On the **Input Message Parsing** tab, set the Message domain property to BLOB.
    d) On the **Advanced** tab:
        - Set the Copy message property to **Copy Entire Message**.
        - Set the Copy local environment property to **Copy Entire LocalEnvironment**.
    e) On the **Request** tab:
        - Set the Input MQMD location property to InputRoot.MQMD.
        - Select the Get by correlation ID property.
    f) On the **Result** tab:
        - Set the Output data location property to OutputLocalEnvironment.ResultRoot,
        - Set the Result data location to ResultRoot.
25. Set the following properties of the Set CorrelId And RTQ Compute node:
    a) On the **Basic** tab, set the Compute mode property to MQ.TASK2.STATE1.
    b) An ESQL module, MQ_Task2_Reply_Set_CorrelId_And_RTQ, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_Reply_Set_CorrelId_And_RTQ module.
    c) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_Reply_Set_CorrelId_And_RTQ
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  SET OutputRoot.MQMD.CorrelId = InputLocalEnvironment.Variables.ResultRoot.BLOB.BLOB;
  SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName =
 InputLocalEnvironment.Variables.ResultRoot.MQMD.ReplyToQ;
  SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueManagername =
 InputLocalEnvironment.Variables.REResultRoot.MQMD.ReplyToQMgr;
  RETURN TRUE;
 END;
END MODULE;
```

    The Set CorrelId And RTQ Compute node:
        i) Extracts the ReplyToQ and ReplyToQMgr from the MQMD of the state message.
        ii) Sets a CorrelId that is the value of the MessageId in the body of the original state message. The CorrelId is used for the Reply message flow.
26. Set the following properties of the Send Reply MQOutput node:
    a) On the **Advanced** tab, set the Destination mode property to Destination list

    By setting the MQOutput node in Destination List mode, the output queue can be set dynamically.
27. Set the following properties of the Handle MQGet Warning Throw node:
    a) On the **Basic** tab, enter error message text into the Message text field. For example, "MQGet returned a warning."

    If a warning occurs in the message from the MQGet node, the Handle_MQGet_Warning Throw node returns an error message about this warning. The message tree contains the MQMD and any message content that was received.

28. Set the following properties of the Handle Error Not Stored Message Throw node:
    a) On the **Basic** tab, enter error message text into the Message text field.
    For example, "MQGet error, no store message queue received."

    If an error occurs because the MQGet node failed to get the message from the store queue, the Handle_Error_Not_Stored_Message Compute node returns an error message. The message is not processed.

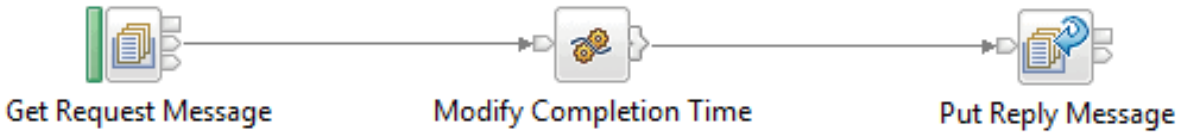
    You cannot return the message from the Reply message flow back to the Request message flow. Therefore, you must consider error handling at a higher level in your messaging architecture than the error handling options that are provided by message flow nodes.

The message flow can now communicate with a backend application. For this scenario, create a third message flow to represent this application.

A backend application, by default, copies the MessageId of the message to its CorrelId, and this value is then sent in the reply message. However, as a real application is not used in this scenario, this action was taken by the Save MQMD Headers to LocalEnvironment Compute node in step "10" on page 756 for the Request message flow.

To demonstrate that the backend application is successfully processed in the Reply message flow, a Compute node modifies the Completion Time value that is in the message.

29. Create a message flow that is named MQ_Task2_BackendApp with the following nodes, and name them on the **Description** tab Node name property:
    - An MQInput node, named Get Request Message.
    - A Compute node, named Modify Completion Time.
    - An MQReply node, named Put Reply Message.
30. Connect the Out terminal of the Get Request Message MQInput node to the In terminal of the Modify Completion Time Compute node.
31. Connect the Out terminal of the Modify Completion Time Compute node to the In terminal of the Put Reply Message MQReply node.



32. Set the following properties of the Get Request Message MQInput node:
    a) On the **Basic** tab, set the Queue name property to MQ.TASK2.BACKEND.REQ1.
    b) On the **MQ Connection** tab, set the Connection property to Local queue manager, and then the Destination queue manager property to TASK2_QUEUE_MANAGER.
    c) On the **Input Message Parsing** tab, set the Message domain property to XMLNSC.
33. Set the following properties of the Modify Completion Time Compute node:
    a) An ESQL module, MQ_Task2_BackendApp_Modify_Completion_Time, is automatically created. In the Application Development view, open **ESQLs**, and then the MQ_Task2_BackendApp_Modify_Completion_Time module.
    b) Copy the following ESQL into the module:

```
CREATE COMPUTE MODULE MQ_Task2_BackendApp_Modify_Completion_Time
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  SET OutputRoot = InputRoot;
  SET OutputRoot.XMLNSC.SaleEnvelope.TimeStamp = CURRENT_GMTTIMESTAMP;
  RETURN TRUE;
 END;
```

```
END MODULE;
```

The `Modify Completion Time` Compute node represents the solution transformation in this scenario. The Compute node adds a time stamp value to the message, which shows the time that it was processed.

34. Set the following properties of the `Get Request Message` MQReply node:
    a) On the **Basic** tab, set the `Queue name` property to `MQ.TASK2.BACKEND.REQ1`.
    b) On the **MQ Connection** tab, set the `Connection` property to `Local queue manager`, and then the `Destination queue manager` property to `TASK2_QUEUE_MANAGER`.
35. Save the message flow.

Results

- When the Request message flow is run, the following actions occur:
  1. The MQInput node reads an IBM MQ request message.
  2. The original message is transformed into an equivalent format for the backend application.
  3. The `ReplyToQ` and `ReplyToQMgr` details are set for the input message to the Reply message flow.
  4. An IBM MQ message is created that contains the transformed message. This message is sent to the backend message flow.
  5. The original `MessageId`, `ReplyToQ`, `ReplyToQMgr` details are saved in a separate IBM MQ message for subsequent retrieval by the Reply message flow.
- When the Reply message flow is run, the following actions occur:
  1. The MQInput reads the IBM MQ message that contains a message in the data format from the backend application.
  2. The message is transformed into the equivalent format for the requester application.
  3. The MQGet node obtains the `CorrelId`, `ReplyToQ`, and `ReplyToQMgr` details of the original request message, by reading theIBM MQ message that was used to store this information in the Request message flow.
  4. An IBM MQ message that contains the transformed message and the retrieved `CorrelId`, `ReplyToQ`, and `ReplyToQMgr` values is created.

How the MQGet node processes messages
The MQGet node processes each message that it receives.

This topic contains the following sections:

Propagating the message

1. If an MQ Message Descriptor header (MQMD) is present in the input tree, the MQGet node uses it. If not, the node creates a default MQMD.
2. The node also creates a default MQ Get Message Options (MQGMO) structure based on the values that you have set for the node properties. If an MQGMO is present in the input tree, the node uses its content to modify the default one.

   When you include an MQGMO to override the default one, you must specify all the options that you are replacing. For example, if you set the option field to MQGMO_CONVERT, that value overrides all options

that you set with the IBM App Connect Enterprise Toolkit. If you do not use an overriding MQGMO, IBM App Connect Enterprise uses the following values:

- If `Wait interval` is not zero, MQGMO_WAIT is set; otherwise, MQGMO_NOWAIT is used.
- If `Transaction mode` is set to `Yes`, MQGMO_SYNCPOINT is used.
- If `Transaction mode` is set to `No`, MQGMO_NOSYNCPOINT is used.
- If `Transaction mode` is set to `Automatic`, MQGMO_SYNCPOINT_IF_PERSISTENT is used.
- The only other option that is used by default in the node properties is MQGMO_COMPLETE_MSG, which is set if `Transaction mode` is set to `Yes` or `No`. This option is not set when your integration node is running on z/OS.
- No other options are used by default.

3. The node makes the `MQGet` call to IBM MQ.
4. The node analyzes the completion code (CC), and propagates the message to the appropriate terminal:

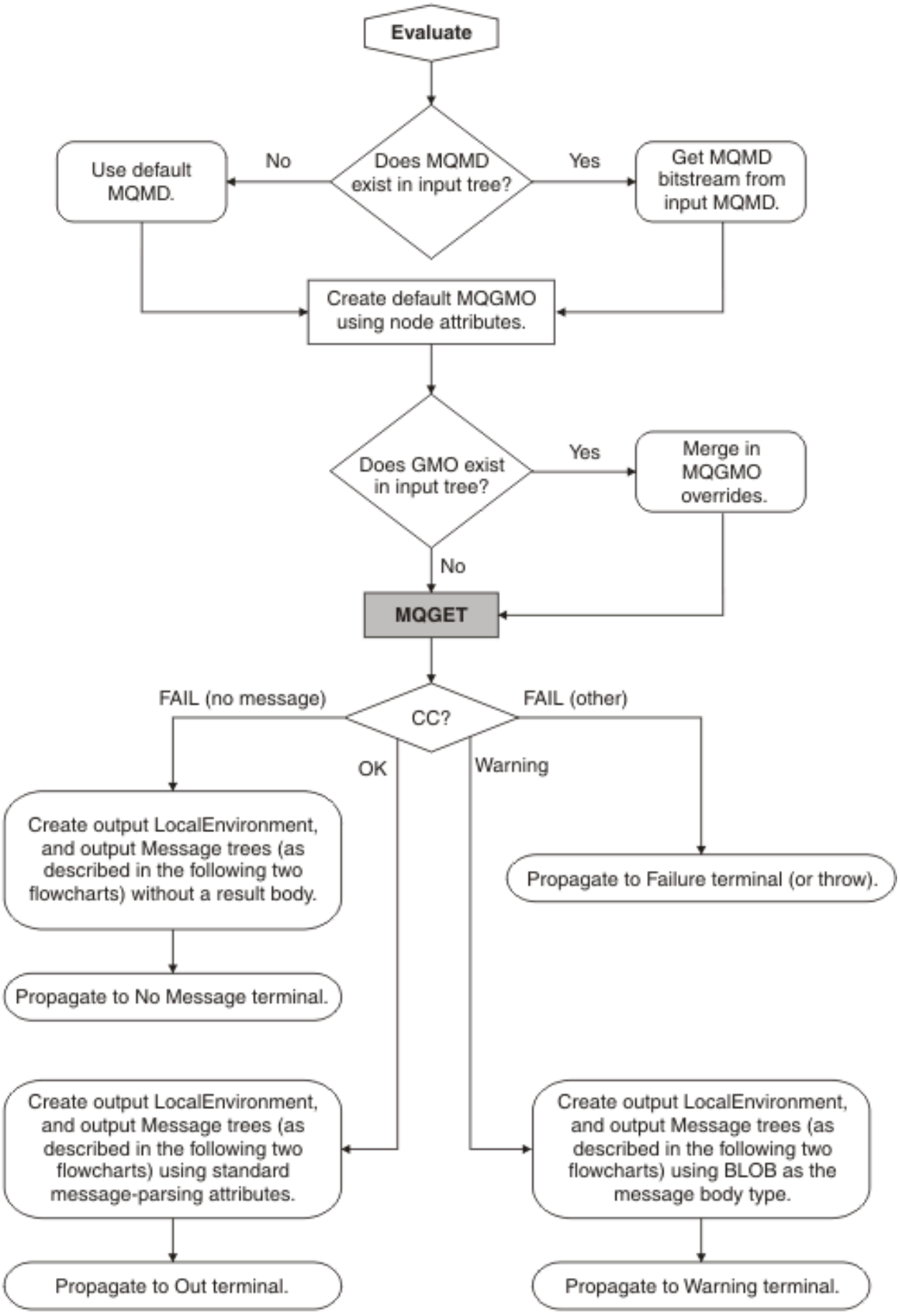
   OK
   The node creates the output LocalEnvironment and the output message trees using standard message-parsing techniques, then propagates the message to the Out terminal.

   Warning
   The node creates the output LocalEnvironment and the output message trees using BLOB as the message body type, then propagates the message to the Warning terminal, if it is connected. If the Warning terminal is not connected, no propagation occurs, and the flow ends.

   Fail (no message)
   The node creates the output LocalEnvironment and the output message trees by copying the input trees, then propagates the message to the No Message terminal, if it is connected. If the No Message terminal is not connected, no propagation occurs. The output message that is propagated to the No Message terminal is constructed from the input message only, according to the values of the `Generate Mode` property, and the `Copy Message` or `Copy Local Environment` properties.

   Fail (other)
   The node propagates the message to the Failure terminal. If the Failure terminal is not connected, the integration node throws an exception and returns control to the closest upstream node that can process it. For more information, see “Handling errors in message flows” on page 1883.

The following diagram shows this processing:

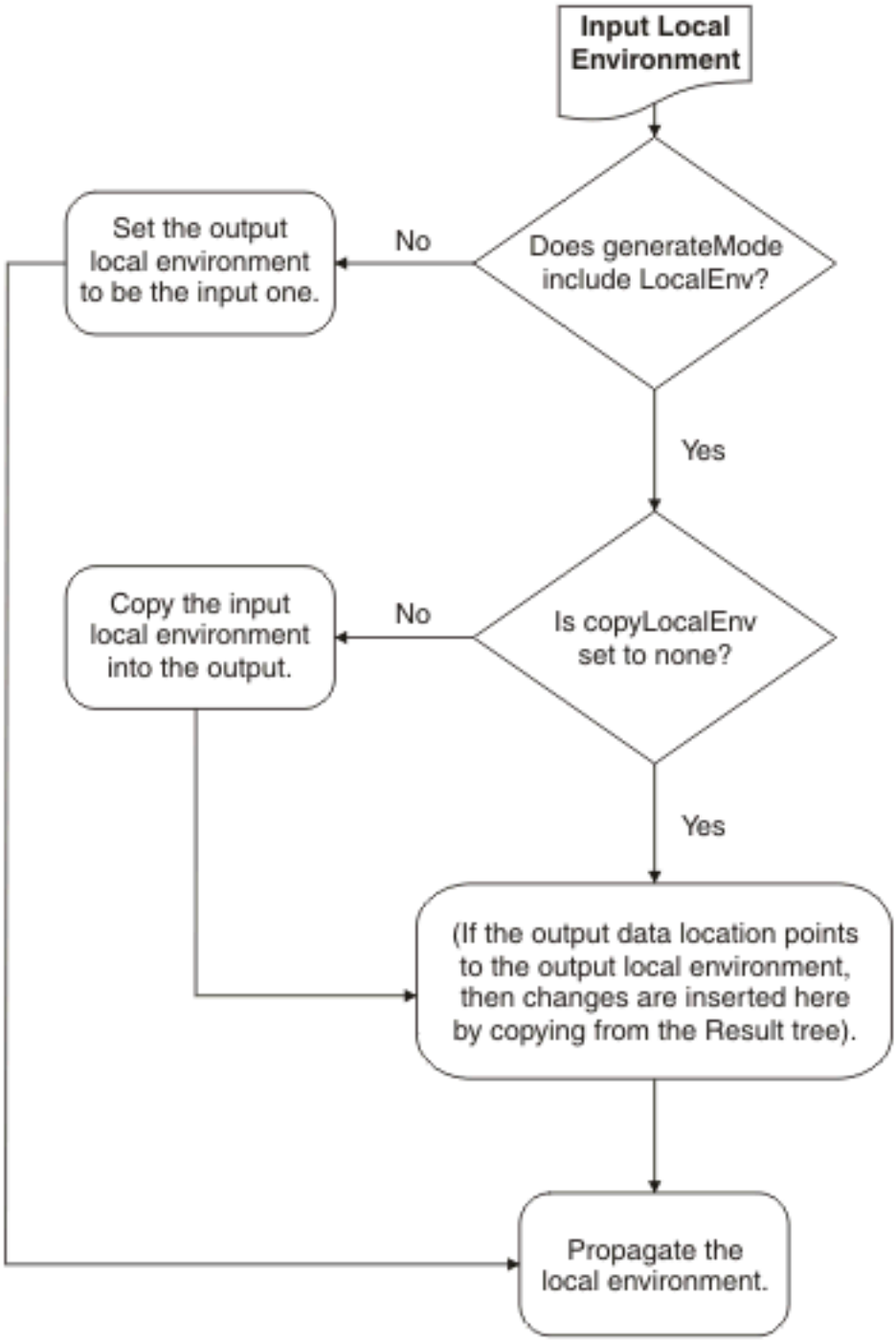

Evaluate
Does MQMD exist in input tree?
No
Use default MQMD.
Yes
Get MQMD bitstream from input MQMD.
Create default MQGMO using node attributes.
Does GMO exist in input tree?
Yes
Merge in MQGMO overrides.
No
MQGET
CC?
FAIL (no message)
FAIL (other)
OK
Warning
Create output LocalEnvironment, and output Message trees (as described in the following two flowcharts) without a result body.
Propagate to Failure terminal (or throw).
Propagate to No Message terminal.
Create output LocalEnvironment, and output Message trees (as described in the following two flowcharts) using standard message-parsing attributes.
Create output LocalEnvironment, and output Message trees (as described in the following two flowcharts) using BLOB as the message body type.
Propagate to Out terminal.
Propagate to Warning terminal.

Constructing OutputLocalEnvironment

1. If the `Generate Mode` property on the MQGet node is set to an option that does not include `LocalEnvironment`, the node copies the input local environment tree to the output local environment tree.

   If this copy is made, any updates that are made in this node to the output local environment tree are not propagated downstream.
2. If the `Copy Local Environment` property is set to an option other than `None`, the node copies the input local environment tree to the output local environment tree.
3. If the output data location points to the output local environment tree, the node applies changes in that tree by copying from the result tree.
4. The local environment tree is propagated.

The following diagram shows this processing:

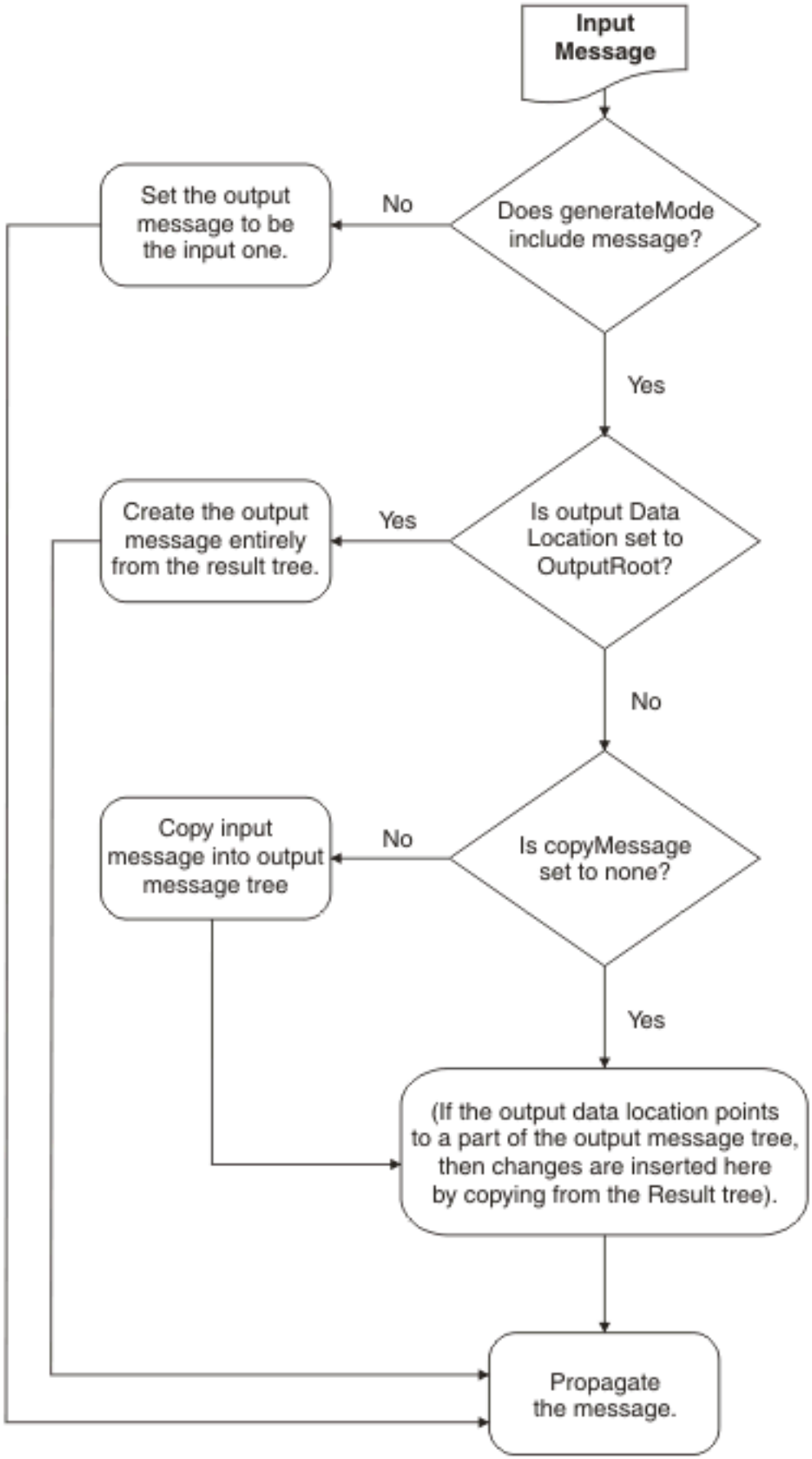


Constructing the Output message

1. If the `Generate Mode` property on the MQGet node is set to an option that does not include `Message`, the node copies the input message tree to the output message tree. Go to step 5.
2. If the `Output Data Location` property is set to `OutputRoot`, the node creates the output message tree entirely from the result tree. Go to step 5.
3. If the `Copy Message` property is set to a value other than `None`, the node copies the input message tree to the output message tree.

4. If the `Output Data Location` property points to a part of the output message tree, the node applies changes in that tree by copying from the result tree at the point that is defined by the `Result Data Location` property.
5. The message tree is propagated.

The following diagram shows this processing:


Input Message
Does generateMode include message?
No
Set the output message to be the input one.
Yes
Is output Data Location set to OutputRoot?
Yes
Create the output message entirely from the result tree.
No
Is copyMessage set to none?
No
Copy input message into output message tree
Yes
(If the output data location points to a part of the output message tree, then changes are inserted here by copying from the Result tree).
Propagate the message.

For an example of how this processing is implemented in a message flow, see "How the MQGet node implements IBM MQ MQGet API calls" on page 768.

How the MQGet node implements IBM MQ MQGet API calls

Review how the MQGet node processes the input messages to construct the output messages, based on both the content of the local environment tree and the input parameters that you set.

You can include an MQGet node anywhere in a message flow, including a flow that implements a request-response scenario. The MQGet node receives an input message on its input terminal from the preceding node in the message flow, issues an MQGET call to retrieve a message from the IBM MQ queue that you have configured in its properties, and builds a result message tree. Finally, it uses the input tree and the result tree to create an output tree that is then propagated to its Output, Warning, or Failure terminal, depending on the configuration of the node and the result of the MQGET operation.

How the MQGet node handles the local environment

The MQGet node examines the local environment tree that is propagated from the preceding node, uses the content that is related to the MQGMO (MQ Get Message Options) and the MQMD (MQ Message Descriptor header), and updates the local environment:

- The node reads the MQGMO structure from `${inputMQParmsLocation}.MQGMO.*`.
- The node copies the IBM MQ completion and reason codes to `${outputMQParmsLocation}.CC` and `${outputMQParmsLocation}.RC`.
- The node writes the complete MQGMO that is used for the MQGET call into `${outputMQParmsLocation}.MQGMO` if `${inputMQParmsLocation}.MQGMO` exists in the input tree.
- The node writes the MQMD that is passed to the MQGET call (that contains the values that are specified in the input message or are generated by the node) into `${inputMQParmsLocation}.MQMD`, deleting any existing content.

Set the value to `${inputMQParmsLocation}` in the MQGet node property Input MQ Parameters Location on the **Request Properties** tab.

Set the value to `${outputMQParmsLocation}` in the MQGet node property Output MQ Parameters Location on the **Result Properties** tab.

For more information about these properties, see node.

In summary:

${inputMQParmsLocation}

- *QueueName*: Optional override for MQGet node *Queue Name* property
- *InitialBufferSize*: Optional override for MQGet node *Initial Buffer Size* property
- *MQGMO.**: Optional MQGET message options that are used by the MQGet node

${outputMQParmsLocation}

- *CC*: MQGET call completion code
- *RC*: MQGET call result code
- *MQGMO.**: MQGET message options that are used if present in `${inputMQParmsLocation}`
- *MQMD*: unparsed MQ Message Descriptor for received messages[1]
- *Browsed*: Set to true if the message is browsed. Not present if the message is removed from the queue

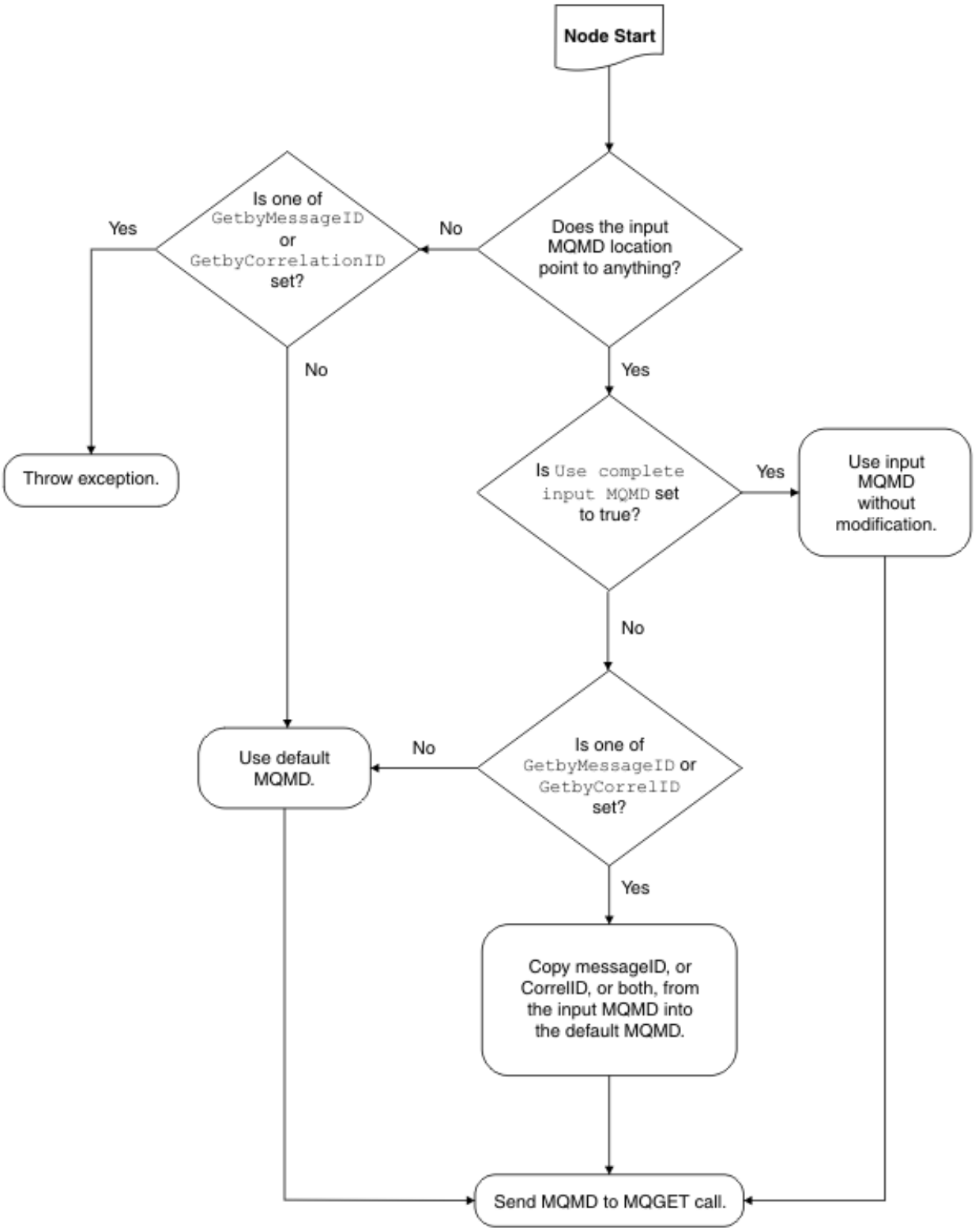
You can parse the MQMD (for example, by using ESQL), where `${outputMQParmsLocation}` is LocalEnvironment.MQ.GET:

```
DECLARE ptr REFERENCE TO OutputLocalEnvironment.MyMQParms;
CREATE FIRSTCHILD OF ptr DOMAIN('MQMD') PARSE(InputLocalEnvironment.MQ.GET.MQMD)
```

How the MQMD for the MQGET call is constructed

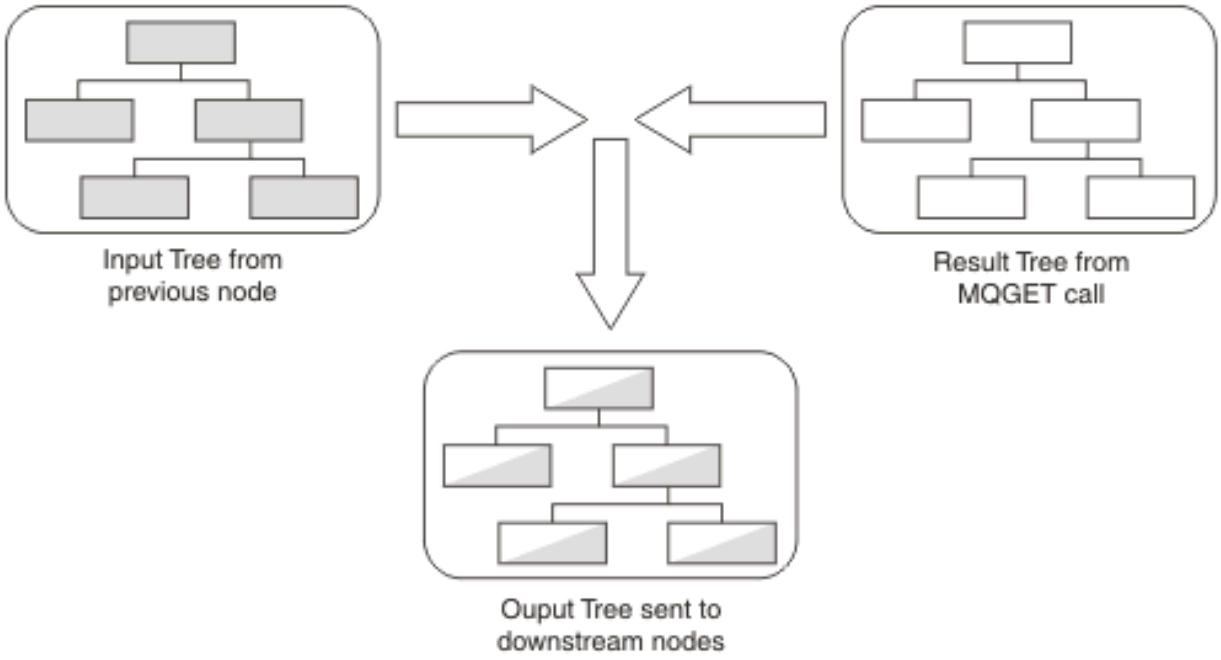
- A default MQMD is prepared. For further information about the MQMD, see the *Application Programming Reference* section of the IBM MQ product documentation online.
- If you do not supply an input MQMD, the default MQMD is used.
- If you do supply an input MQMD, the default MQMD is used after the following modifications:
  - If the property `Use all input MQMD fields` is set, all MQMD fields supplied are copied into the default MQMD from the input MQMD.
  - If the property `Use all input MQMD fields` is not set and the properties `Get by Message ID` or `Get by Correlation ID` are selected, the respective IDs are copied into the default MQMD from the input MQMD.

The following diagram shows how the MQGet node constructs the MQMD that is used on the call to IBM MQ:



How the output message tree is constructed

The following diagram outlines how the MQGet node constructs the output message tree, combining the input tree from the previous node with the result tree from the MQGET call:



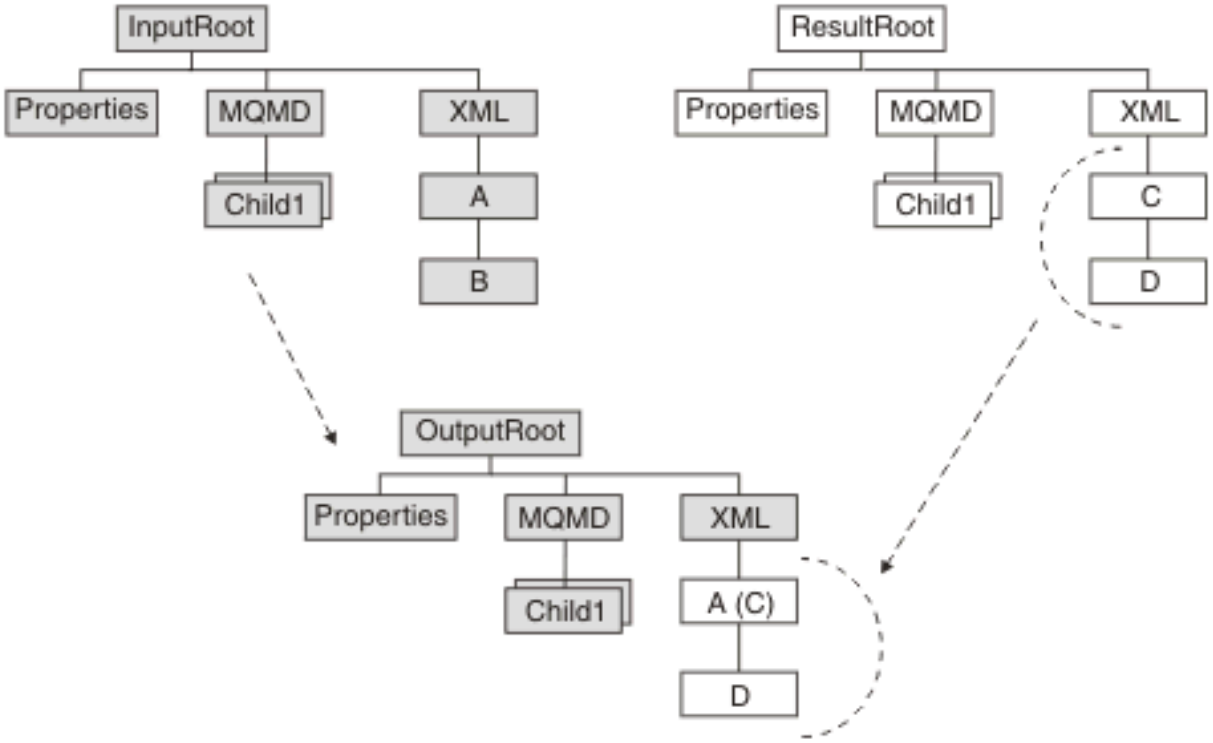
In this example, the MQGet node properties are configured as shown in the following table.

Property	**Action**
Copy Message	Copy Entire Message
Generate Mode	Message
Output Data Location	`OutputRoot.XMLNS.A`
Result Data Location	`ResultRoot.XMLNS.C`

The MQGet node constructs the output tree according to the following sequence:

1. The whole of the input tree is copied to the output tree, including the XML branch with child A, and A's child B.
2. From the result tree, the XML branch's child C, and C's child D, are put into the output tree at position `OutputRoot.XMLNS.A`. Any previous content of A (values and children) is lost, and replaced with C's content, including all values and children it has, in this case child D.
3. The position in the output tree retains the name A.

The following diagram shows this behavior:



For some examples of message trees that are constructed by the MQGet node according to the rules described above, see “MQGet node message tree examples” on page 772.

MQGet node message tree examples

The MQGet node generates message trees based on the input message assembly that it receives, and the options that you have set on the node properties.

The message trees, shown in the following table, are generated according to the rules described in “How the MQGet node implements IBM MQ MQGet API calls” on page 768.

With a message assembly like this:	The message that the MQGet node returns is:
InputRoot **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **InputLocalEnvironment** **MQ** **GET** **MQGMO** MatchOptions = MQMO_MATCH_CORREL_ID **MQMD (with no children)** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary})	**ResultRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Input MQMD Location InputLocalEnvironment.Variables.MQMD **Copy Message** Copy Entire Message **Copy Local Environment** Copy Entire LocalEnvironment **Generate Mode** Message and LocalEnvironment **Output Data Location** InputLocalEnvironment.Variables.ReturnedMessage	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XML** {result message body}

With the following node property settings:	The resulting output message assembly is:
Result Data Location ResultRoot.XML	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree is effectively the result of doing an assignment from `${resultDataLocation}` to `${outputDataLocation}`. The value of the source element is copied, as are all children including attributes.

With the following node property settings:	The resulting output message assembly is:
Copy Local Environment None	**OutputRoot** **MQMD** {input message MQMD} **MQRFH2** {input message MQRFH2} **XMLNS** {input message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **ReturnedMessage (with any attributes and value from ResultRoot.XML)** {result message body} This tree has the MQMD that is used for the MQGET call in the OutputLocalEnvironment, because the input MQ parameters location had an MQMD element under it. Even though the input tree is not copied, the presence of the MQMD element causes the MQMD that is used for the MQGET call to be placed in the output tree.

With the following node property settings:	The resulting output message assembly is:
Output Data Location <blank> **Copy Local Environment** Copy Entire Local Environment	**OutputRoot** **MQMD** {result message MQMD} **MQRFH2** {result message MQRFH2} **XMLNS** {result message body} **OutputLocalEnvironment** **MQ** **GET** **MQGMO** {MQGMO used for MQGET} **MQMD** {MQMD used for MQGET} **CC = 0** **RC = 0** **Variables** **MQMD** {input MQMD} (with CorrelID = {correct Correlation ID as binary}) The value that you set for the Copy Message property makes no difference to the eventual output tree in this case.

Enabling IBM MQ applications

If you want to connect IBM MQ applications to the integration node, you must define and secure the required resources.

About this task

- "Defining IBM MQ resources" on page 777
- "Securing IBM MQ resources" on page 778

Defining IBM MQ resources

An application client can run on a computer anywhere in the IBM MQ network. If your applications use IBM MQ facilities to connect to the integration node, and to interact with it (by using the MQI and AMI), you must set up the IBM MQ resources that they require.

About this task

The way that you set up applications is identical to the setup for clients for an IBM MQ server. To support client connections to an integration node:

Procedure

1. If the application runs on the same computer as the integration node, it can establish a local connection with the integration node queue manager by using MQCONN, and you do not have to define any IBM MQ resources to support it.

2. If the application runs on the same computer as another queue manager in the IBM MQ network, it can establish a local connection to that queue manager. In this scenario, you must define the appropriate resource to support communications between the queue manager to which the client has connected and the queue manager that hosts the integration node that provides the required service.
3. If the application runs on a computer that does not support a queue manager, it must make a client connection to a queue manager on another computer:
   - The integration node queue manager

     You must set up the appropriate client connection and server connection definitions to support this option.
   - Another queue manager in the network

     You must set up the appropriate client connection and server connection definitions to support this option, and ensure that definitions are in place to support communications between the queue manager to which the client has connected, and the queue manager that hosts the integration node that provides the required service.

Results

An application can get messages only from queues that are owned by the queue manager to which it is connected (this restriction is true for all IBM MQ applications). Therefore, if an application expects to receive messages from a queue populated by a service within a particular integration node and owned by that integration node queue manager, it must connect to that integration node queue manager (by using a local or IBM MQ client connection).

An application that puts messages, however, can be connected to any queue manager in the IBM MQ network, if the queue manager can resolve the target destination in some way. In all cases, the queue manager to which the client application is connected must know the location of the queue or queues to which the application puts messages (for example, by using remote queue definitions).

When you define an IBM MQ queue for a node in a message flow, you must not give it a name that starts with SYSTEM_BROKER. Names that include these characters are reserved for queues that are defined for internal use by IBM App Connect Enterprise components.

If your application is a subscriber that receives messages published by other applications, it can specify a temporary dynamic queue as its subscriber queue. If it does so, the integration node automatically deregisters the subscription when the queue is deleted.

For more details about applications, putting and getting messages, and the use of IBM MQ clients, see the *Clients* and *Application Programming Guide* sections of the IBM MQ product documentation online.

Securing IBM MQ resources
Secure the IBM MQ resources that your integration solution requires.

About this task

This information does not apply to z/OS.

Integration servers depend on a number of IBM MQ resources to operate successfully. You must control access to these resources to ensure that the integration servers can access the resources on which they depend, and that these same resources are protected from other users.

You can configure a connection to a secured IBM MQ queue manager, by setting properties on an MQ node or in an policy. If you are connecting to a local queue manager, you can optionally configure the connection to use a security identity for authentication. If you are configuring a client connection to a remote queue manager, you can configure the connection to use a security identity for authentication, SSL for confidentiality, or both. For more information, see “Connecting to a secured IBM MQ queue manager” on page 747.

Some authorizations are granted on your behalf when commands are issued. Others depend on the configuration of your integration server network.

- When you start the IBM App Connect Enterprise Toolkit, it connects to an integration server by using an IBM MQ local or client connection. For more information about IBM MQ channel security, see Setting up IBM MQ MQI client security in the IBM MQ product documentation online.
- When you create and deploy a message flow that includes nodes which reference IBM MQ queues, grant get, inq, and put authority to the user ID under which the integration server is running.

Application programming interfaces

IBM App Connect Enterprise supports programming interfaces that are in use by IBM MQ applications; it does not provide any unique programming interfaces.

For example, you can use the Message Queue Interface (MQI). The MQI provides a few calls that allow an application to interact with other applications in a network of IBM MQ queue managers. The calls support a large range of parameters that allow a rich choice of processing options for each message.

Client applications that use the MQI can run on any supported IBM MQ operating system, and therefore any limitations that are enforced for language or function are defined by the relevant product for that operating system.

The MQI is described in the *Application Programming Reference* and *Application Programming Guide* sections in the IBM MQ product documentation online. Details are also provided of the programming language and operating system support available for clients that use this interface.

If you have existing user applications that are written for an IBM MQ interface, the application can typically run unchanged in an integration node environment. You must create the message flows to interact with these applications from the supported protocols, by using the appropriate input and output nodes. IBM App Connect Enterprise provides built-in input and output nodes for its supported protocols and you can create your own user-defined nodes to support additional protocols.

You can also create new user applications to interact with the integration node.

Message headers

IBM App Connect Enterprise provides parsers for many IBM MQ headers, and can therefore accept messages that contain these headers from the IBM MQ Enterprise Transport protocol.

Messages must include an IBM MQ Message Descriptor (MQMD) as the first header, which must precede user or application data in every message. The MQMD contains basic control information that must travel with the message, including:

- The message identifier
- The destination of the reply, if one is to be sent
- Reply and report options (for example, confirm on delivery report)
- The format of any following data in the message

When a message is processed by an IBM App Connect Enterprise integration node, it typically (but not necessarily) has one or more additional headers. The header that follows the MQMD is always identified in the format field within the MQMD, and itself contains another format field to identify either the header that follows, or the format of the user data.

The additional headers can include:

MQRFH
The Rules and Formatting header is used by IBM MQ Publish/Subscribe.

MQRFH2
The MQRFH2 is an updated version of MQRFH and allows Unicode strings to be transported without translation, and it can carry numeric data types. The MQRFH2 header carries a description of the message contents, so that IBM App Connect Enterprise can select the correct message parser when content-based processing is carried out on the message. In addition, this header contains publish/subscribe command messages.

Use the MQRFH2 header in all new applications that are written for the IBM App Connect Enterprise environment that use a supported protocol that is based on IBM MQ technology. The MQRFH2 header must be immediately before the body of the message (that is, the last header).

If an MQRFH2 header is not included (which is normally the case of the application uses a supported protocol that is not based on IBM MQ technology), you must configure the message flow that processes its messages to specify the message characteristics (by setting the input node properties).

Using IBM MQ cluster queues for input and output

Design your integration node network to use IBM MQ queues, if appropriate for your business needs.

About this task

The use of queue manager clusters has the following significant benefits:

1. Reduced system administration

   Clusters need fewer definitions to establish a network; you can set up and change your network more quickly and easily.

2. Increased availability and workload balancing

   You can benefit by defining instances of the same queue to more than one queue manager, therefore distributing the workload through the cluster.

If you use clusters with IBM App Connect Enterprise, consider the following queues:

For SYSTEM.BROKER queues:
The SYSTEM.BROKER queues are defined for you when you create IBM App Connect Enterprise components, and are not defined as cluster queues. Do not change this attribute.

For message flow input queues:
If you define an input queue as a cluster queue, consider the implications for the order of messages or the segments of a segmented message. The implications are the same as for any IBM MQ cluster queue. In particular, the application must ensure that, if it is sending segmented messages, all segments are processed by the same target queue, and therefore by the same instance of the message flow at the same integration node.

For message flow output queues:

- IBM App Connect Enterprise always specifies MQOO_BIND_AS_Q_DEF when it opens a queue for output. If you expect segmented messages to be put to an output queue, or want a series of messages to be handled by the same process, you must specify DEFBIND(OPEN) when you define that queue. This option ensures that all segments of a single message, or all messages in a sequence, are put to the same target queue and are processed by the same instance of the receiving application.
- If you create your own output nodes, specify MQOO_BIND_AS_Q_DEF when you open the output queue, and DEFBIND(OPEN) when you define the queue, if you need to ensure message order, or to ensure a single target for segmented messages.

For publish/subscribe applications:

- If the target queue for a publication is a cluster queue, you must deploy the publish/subscribe message flow to all the integration nodes on queue managers in the cluster. However, the cluster does not provide any of the failover function to the integration node network and function. If an integration node to which a message is published, or a subscriber registers, is unavailable, the distribution of the publication or registration is not taken over by another integration node.
- When a client registers a subscription with an integration node that is running on a queue manager that is a member of a cluster, the integration node forwards a proxy registration to its neighbors in the integration node domain; the registration details are not advertised to other members of the cluster.
- A client might choose to become a clustered subscriber, so that its subscriber queue is one of a set of clustered queues that receive a particular publication. In this case, when registering a

subscription, use the name of an "imaginary" queue manager that is associated with the cluster; this queue manager is not the one to which the publication is sent, but an alias for the integration node to use. As an administrative activity, a blank queue manager alias definition is made for this queue manager on the integration node that satisfies this subscription for all clustered subscribers. When the integration node publishes to a subscriber queue that names this queue manager, resolution of the queue manager name results in the publication being sent to any queue manager that hosts the subscriber cluster queue, and only one clustered subscriber receives the publication.

For example, if the clustered subscriber queue was SUBS_QUEUE and the "imaginary" subscriber queue manager was CLUSTER_QM, the integration node definition is:

```
DEFINE QREMOTE(CLUSTER_QM) RQMNAME(' ') RNAME(' ')
```

This configuration sends integration node publications for SUBS_QUEUE on CLUSTER_QM to one instance of the cluster queue named SUBS_QUEUE anywhere in the cluster.

To understand more about clusters, and the implications of using cluster queues, see Queue manager clusters in the IBM MQ product documentation online.

You can also configure IBM App Connect Enterprise to use queues on an IBM MQ uniform cluster. A uniform cluster is a specific pattern of an IBM MQ cluster that provides a highly available and horizontally scaled small collection of queue managers. In scenarios in which one-way messaging is required, it can be useful to use queues that are defined on an IBM MQ uniform cluster. For more information, see “Using queues on an IBM MQ uniform cluster” on page 781.

Using queues on an IBM MQ uniform cluster

You can configure IBM App Connect Enterprise to use queues on an IBM MQ uniform cluster.

About this task

A uniform cluster is a specific type of IBM MQ cluster that provides a highly available and horizontally scaled small collection of queue managers. These queue managers are configured almost identically, so that an application can interact with them as a single group. This approach makes it easier to ensure that each queue manager in the cluster is being used, by automatically ensuring that application instances are spread evenly across the queue managers.

Uniform clusters differ from other IBM MQ clusters in several respects. For example, a uniform cluster typically has a smaller number of queue managers in the cluster and is used by a single application or a group of related applications. In a uniform cluster, client connections are grouped together based on the application name. Applications that connect to any member of the uniform cluster using the same application name are considered to be equivalent to any other applications using the same application name. All connections from an App Connect Enterprise integration server currently use the same MQ application name.

By using uniform clusters, applications can be designed for scale and availability, and can connect to any of the queue managers within the uniform cluster. As a result, there is no dependency on a specific queue manager, which improves availability and workload balancing of messaging traffic. For more information, see Uniform clusters in the IBM MQ product documentation online.

In some scenarios, it is important that the connection that is made between an IBM App Connect Enterprise integration node or server and an IBM MQ queue manager is maintained with that same queue manager, and not changed to an alternative queue manager. In these circumstances, uniform clusters are not appropriate. For example, in some MQ request/reply scenarios, reply messages must be returned to the same App Connect Enterprise integration server, which can be problematic if the connection is moved to a different queue manager. Another situation in which it is necessary to maintain access to a specific queue manager is when a flow contains nodes that require access to information about the state of in-flight messages, such as Aggregation, Sequence, Resequence, Collector, and Timeout nodes. Information about the state of in-flight messages is held on storage queues that are owned by the queue manager that is associated with the integration server. If the connection is moved to a different queue manager, the required information will no longer be accessible. Uniform clusters are also unsuitable for connections that are made from locally-bound applications; uniform clusters cannot achieve even

workload distribution with locally bound applications, because they cannot connect to any other member of the cluster. For more information, see Limitations and considerations for uniform clusters.

In scenarios where one-way messaging is required, it can be beneficial to use queues that are defined on an IBM MQ uniform cluster. For information about configuring App Connect Enterprise to use uniform clusters, see "Configuring App Connect Enterprise to use IBM MQ uniform clusters" on page 782.

Configuring App Connect Enterprise to use IBM MQ uniform clusters
You can configure IBM App Connect Enterprise to use queues on an IBM MQ uniform cluster.

About this task

A uniform cluster is a specific pattern of an IBM MQ cluster that provides a highly available and horizontally scaled small collection of queue managers. In scenarios in which one-way messaging is required, it can be useful to use queues that are defined on an IBM MQ uniform cluster. For more information about IBM MQ uniform clusters, see "Using queues on an IBM MQ uniform cluster" on page 781.

When messages are running through a non-transactional message flow, if the connection is moved part of the way through the flow to another queue manager in the uniform cluster, the flow continues to operate. Messages that are sent before and after the reconnection might be delivered to different queue managers. This can occur even for separate message flow node instances in the same flow.

In a transactional message flow, if the connection is moved part of the way through the flow to another queue manager in the uniform cluster, the transaction is rolled back and an exception is thrown from the next MQ call, and the backed out message is still on the original queue manager. When the connection has been re-established, the backed out message is available to be processed again, typically in another instance of the application. Optionally, you can catch and ignore the exception, at which point your message flow can continue to do more work on the re-established connection, committing only the work after the connection was re-established.

If a connection is moved to another queue manager in the uniform cluster when a message flow is idle (polling on MQGET without receiving a message), the integration server reconnects to the new queue manager and then continues to poll.

The following sections explain how you can configure IBM MQ to connect App Connect Enterprise to a uniform cluster, and how you can configure App Connect Enterprise to use a connection to an IBM MQ uniform cluster, by using an MQEndpoint policy:

- "Configuring IBM MQ to connect App Connect Enterprise to a uniform cluster" on page 782
- "Configuring IBM App Connect Enterprise to connect to a uniform cluster" on page 782

Configuring IBM MQ to connect App Connect Enterprise to a uniform cluster

About this task

You can connect IBM App Connect Enterprise to an IBM MQ uniform cluster by configuring the connection in IBM MQ. To configure the connection in IBM MQ, configure Client Channel Definition Table (CCDT), and then set either the MQCLNTCF environment variable or configure the `mqclient.ini` file to control the MQ Client code. For more information, see the IBM MQ product documentation online.

Configuring IBM App Connect Enterprise to connect to a uniform cluster

About this task

You can configure App Connect Enterprise to use a connection to an IBM MQ uniform cluster by using the `Reconnect option` property on the MQEndpoint policy. This property enables you to control whether the IBM MQ client automatically attempts to reconnect to the queue manager if the connection is lost. By default, the IBM MQ client's default reconnection strategy is used, modified by using the IBM MQ client configuration described in "Configuring IBM MQ to connect App Connect Enterprise to a uniform cluster" on page 782.

The Reconnect option property is used only for client connections and is ignored for server connections. The property takes the following possible values:

- default - This value corresponds to the IBM MQ option MQCNO_RECONNECT_AS_DEF. This is the default value for the property.
- enabled - This value corresponds to the IBM MQ option MQCNO_RECONNECT.
- disabled - This value corresponds to the IBM MQ option MQCNO_RECONNECT_DISABLED.
- queueManager - This value corresponds to the IBM MQ option MQCNO_RECONNECT_Q_MGR.

These values are passed to the IBM MQ client in the MQCONNX call, which is made by an MQInput node at the start of a message flow. This type of configuration enables you to use different reconnection settings for different deployments to the same integration server, which can be configured to use different MQEndpoint policies. For more information, see policy.

As a result of the mappings between the App Connect Enterprise reconnect settings and the MQCNO options, you must ensure that the options that you select on the MQ nodes do not conflict with your chosen MQCNO option. For example, you cannot use the **Logical order** property on the MQInput node (which is set by default) in conjunction with setting the **Reconnect option** property in the MQEndpoint policy to enabled (which corresponds to the IBM MQ option MQCNO_RECONNECT).

Configuring flows to handle IBM MQ message groups

IBM MQ allows multiple messages to be treated as a group, or as segments of one larger message. IBM App Connect Enterprise provides support for IBM MQ message grouping and partial support for message segmenting.

You can use the MQInput and MQOutput nodes to receive and send messages that are part of an IBM MQ message group. You can use the MQOutput node to send messages that are segments of a larger message.

For guidance about configuring the MQInput and MQOutput nodes to receive and send messages that are part of an IBM MQ message group, see:

- “Receiving messages in an IBM MQ message group” on page 783
- “Sending messages in an IBM MQ message group” on page 785
- “Sending message segments in an IBM MQ message” on page 786

For more information about IBM MQ message groups, see Message groups in the IBM MQ product documentation online.

Receiving messages in an IBM MQ message group

You can configure the MQInput node to receive messages that are in an IBM MQ message group.

About this task

The following properties on the MQInput node control the processing of messages in an IBM MQ message group:

- Logical order
- Order mode
- All messages available
- Commit by message group

To ensure that your message flow receives group messages in the order that has been assigned by the sending application, select Logical order. If you do not select this option, messages that are sent as part of a group are not received in a predetermined order. This property maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the IBM MQ product documentation online.

If you specify a value of By Queue Order on the Order mode property, the message flow processes the messages in the group in the order that is defined by the queue attributes; this order is guaranteed to be

preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the `Additional instances` property is set to zero. The message flow processes the messages on a single thread of execution, and a message is processed to completion before the next message is retrieved from the queue. If you do not specify this value, it is possible that multiple threads within a single message flow are processing multiple messages, and the final message in a group, which prompts the commit or roll back action, is not guaranteed to be processed to completion after all other messages in the group.

To ensure that only a single instance of the message flow processes the group messages in the order that has been assigned by the sending application, select `Logical order` **and** specify a value of `By Queue Order` on the `Order mode` property.

If you select `All messages available`, message retrieval and processing is performed only when all messages in a single group are available. This means that messages in a group are not received until all the messages in the group are present on the input queue. It is good practice to select this check box when your message flow needs to process group messages. If you do not select this check box, the message flow receives the messages as they arrive on the input queue; if a message in the group fails to arrive on the input queue, the message flow waits for it and cannot process any further messages until this message arrives. This property maps to the MQGMO_ALL_MESSAGES_AVAILABLE option of the MQGMO of the MQI. More information about the options to which this property maps is available in the *Application Programming Reference* section of the IBM MQ product documentation online.

If you select `Commit by message group`, message processing is committed only after the final message of a group has been received and processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow. This property is relevant only if you have selected `Logical order`. It is good practice to select this check box together with the `All messages available` check box because this ensures that the complete message group is retrieved and processed in the same unit of work without risk of the message flow waiting indefinitely for messages in the group to arrive on the input queue.

To ensure that the message flow that processes group messages does not wait for unavailable messages:

- Avoid having multiple message flows reading from the same input queue when group messages are being retrieved.
- Avoid deploying additional instances of a flow that retrieves group messages.
- Avoid using expired messages in message groups.
- When expired messages are to be used, ensure either that all messages have the same expiry time or that the first message in the group is set to expire before the rest of the group. If the first message in a group cannot be retrieved, the group can never be started in logical order.

If a message flow waits for a group message that does not arrive within the wait interval, a BIP2675 warning message is issued. From that point on, the message flow always attempts to retrieve the next group message and does not process any other input messages until the next group message is received.

Therefore, if the expected group message does not arrive, or has expired, the message flow must be stopped manually, and any incomplete message group cleared from the input queue.

A message flow cannot receive all the messages in a group in one operation.

If you specify a value of `Yes` or `No` on the `Transaction mode` property, all the segments in a message are received in the message flow as a single message. As a result, the message flow might receive very large messages which might lead to storage problems in the integration node. If you specify a value of `Automatic` on the `Transaction mode` property, message segments are received as individual messages.

Sending messages in an IBM MQ message group
The MQOutput node can send multiple messages that form an IBM MQ message group. Configure a Compute or JavaCompute node to set the MQMD fields to specify message group options.

About this task

The message flow must set the following MQMD fields:

- **GroupId**
- **MsgSeqNumber**
- **MsgFlags**

You can use the following example of ESQL code in a Compute node, which shows how to set these fields:

```
DECLARE MSGNUM INT 0;
  DECLARE MSGTOTAL INT 5;
  WHILE MSGNUM < MSGTOTAL DO
    SET MSGNUM = MSGNUM + 1;
    CALL CopyMessageHeaders();
    -- Manually set the groupId since we cant ask the queue manager to generate one.
    -- the UUIDASBLOB function could be used here to generate one, but this must be done
    -- outside the loop to keep the same groupId throughout!
    SET OutputRoot.MQMD.GroupId = X'000000000000000000000000000000000000000000000001';
    SET OutputRoot.MQMD.MsgSeqNumber = MSGNUM;
    SET OutputRoot.MQMD.MsgFlags = MQMF_MSG_IN_GROUP;
    IF (MSGNUM = MSGTOTAL) THEN
      SET OutputRoot.MQMD.MsgFlags = MQMF_LAST_MSG_IN_GROUP;
    END IF;
    SET OutputRoot.XML.TestCase = MSGNUM;
    PROPAGATE;
  END WHILE;
  RETURN FALSE;
```

You can use the following example of Java code in a JavaCompute node, which shows how to set these fields:

```
public class mq_group_java_JavaCompute extends MbJavaComputeNode {
      public void evaluate(MbMessageAssembly assembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        int MSGTOTAL = 5;
        for(int MSGNUM = 1 ; MSGNUM <= MSGTOTAL;MSGNUM++){
           MbMessage newMessage = new MbMessage();
           MbElement root = newMessage.getRootElement();
           MbElement mqmd = root.createElementAsFirstChild("MQMD");
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
           "GroupId", new byte[]{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1});
           mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "MsgSeqNumber", MSGNUM);
           if (MSGNUM == MSGTOTAL){
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 16); // MQMF_LAST_MSG_IN_GROUP 0x00000010
           }
           else{
               mqmd.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
               "MsgFlags", 8);  // MQMF_MSG_IN_GROUP      0x00000008
           }
           MbElement xmlnsc = root.createElementAsLastChild("XMLNSC");
           MbElement body =
 xmlnsc.createElementAsLastChild(MbElement.TYPE_NAME,"TestCase",null);
           body.setValue(MSGNUM);
           MbMessageAssembly newAssembly = new MbMessageAssembly(assembly,newMessage);
           out.propagate(newAssembly);
        }
    }
}
```

If the message flow is sending multiple messages from one input message, it can create a **GroupId** value, increment the **MsgSeqNumber** value, and set the **MsgFlags** field. The examples of ESQL and Java code show how you can set these values. However, if the message flow is sending multiple messages from more than one input message, it must store the **GroupId** and **MsgSeqNumber** values between flow instances, which is achieved by using shared variables.

For more information about message grouping, see Message groups in the IBM MQ product documentation online. For more information about IBM MQ fields, see MQMD - Message descriptor of the IBM MQ product documentation online.

Sending message segments in an IBM MQ message
The MQOutput node can send multiple message segments that form an IBM MQ message. Configure a Compute node to set the MQMD fields to specify message segment options.

About this task

You can either select `Segmentation allowed` on the node, or set the required fields in the MQMD in the message flow:

- GroupId
- MsgFlags
- Offset

Use the example ESQL code in "Sending messages in an IBM MQ message group" on page 785 and change the code to set these fields.

For more information about message grouping and segmentation, see Message groups of the IBM MQ product documentation online.

Ensuring that messages are not lost

Messages that flow through your integration node domain represent business data that you want to safeguard. Configure the messages, your environment, or both, to ensure that you do not lose messages.

About this task

Messages that are generated by your applications and by runtime components for inter-component communication are important to the operation of your integration nodes.

For messages traveling across IBM MQ, two techniques can protect against message loss:

- Message persistence

  If a message is persistent, IBM MQ ensures that it is not lost when a failure occurs, by copying it to disk.
- Sync point control

  An application can request that a message is processed in a synchronized unit-of-work (UOW)

For more information about how to use these options, refer to the *System Administration Guide* section of the IBM MQ product documentation online.

If delivery of application messages is critical, you must design application programs and the message flows that they use to ensure that messages are not lost. The techniques used depend on the protocol used by the applications.

IBM MQ Enterprise Transport

If you are using the built-in input nodes that accept messages across the IBM MQ protocol, you can use the following guidelines and recommendations:

- Using persistent messages

  IBM MQ messaging products provide *message persistence*, which defines the longevity of the message in the system and guarantees message integrity. Nonpersistent messages are lost in the

event of system or queue manager failure. Persistent messages are always recovered if a failure occurs.

You can control message persistence in the following ways:

- Program your applications that put messages to a queue using the MQI or AMI to indicate that the messages are persistent.
- Define the input queue with message persistence as the default setting.
- Configure the output node to handle persistent messages.
- Program your subscriber applications to request message persistence.

When an input node reads a message from an input queue, the default action is to use the persistence defined in the IBM MQ message header (MQMD), that has been set either by the application creating the message, or by the default persistence of the input queue. The message retains this persistence throughout the message flow, unless it is changed in a subsequent message processing node.

You can override the persistence value of each message when the message flow terminates at an output node. This node has a property that allows you to specify the message persistence of each message when it is put to the output queue, either as the required value, or as a default value. If you specify the default, the message takes the persistence value defined for the queues to which the messages are written.

If a message passes through a Publication node, the persistence of messages sent to subscribers is determined by the registration options of the subscribers. If a subscriber has requested persistent message delivery, and is authorized to do so by explicit or implicit (inherited) ACL, the message is delivered persistently regardless of its existing persistence property. Also, if the user has requested nonpersistent message delivery, the message is delivered nonpersistent regardless of its existing persistence property.

If a message flow creates a message (for example, in a Compute node), the persistence in the MQMD of the new message is copied from the persistence in the MQMD of the incoming message.

- Processing messages under sync point control

  The default action of a message flow is to process incoming messages under sync point in a transaction that is controlled by the integration node. This means that a message that fails to be processed for any reason is backed out by the integration node. Because it was received under sync point, the failing message is reinstated on the input queue and can be processed again. If the processing fails, the error handling procedures that are in place for this message flow (defined either by how you have configured the message flow, or by the integration node) are executed.

IBM MQ Web Services Transport
If you are using the HTTPInput, HTTPRequest, SOAPInput, SOAPRequest nodes, or a SOAPAsyncRequest and SOAPAsyncResponse node pair that accept messages from web services applications, no facilities are available to protect against message loss. You can, however, provide recovery procedures by configuring the message flow to handle its own errors.

Other transports and protocols
If you have created your own user-defined input nodes that receive messages from another transport protocol, you must rely on the support provided by that transport protocol, or provide your own recovery procedures.

Processing HTTP messages

Hypertext Transfer Protocol (HTTP) is an Internet protocol that is used to transfer and display hypertext and XML documents on the Web.

You can configure message flows that include the HTTP or SOAP nodes to access the HTTP transport to work with the following resources:

- SOAP-based web services
- Other web services standards, such as REST or XML-RPC

- General HTTP messaging, where the payload might be XML

HTTP nodes can process non-secure (HTTP) messages and secure (HTTPS or HTTP over SSL) messages.

For SOAP-based web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

For more information about using SOAP messages and message flow nodes, see "What is SOAP?" on page 807

You can choose how your HTTP and SOAP nodes interact with the TCP/IP network:

- You can use the integration node listener, which receives HTTP messages on one port, and HTTPS messages on a second port.

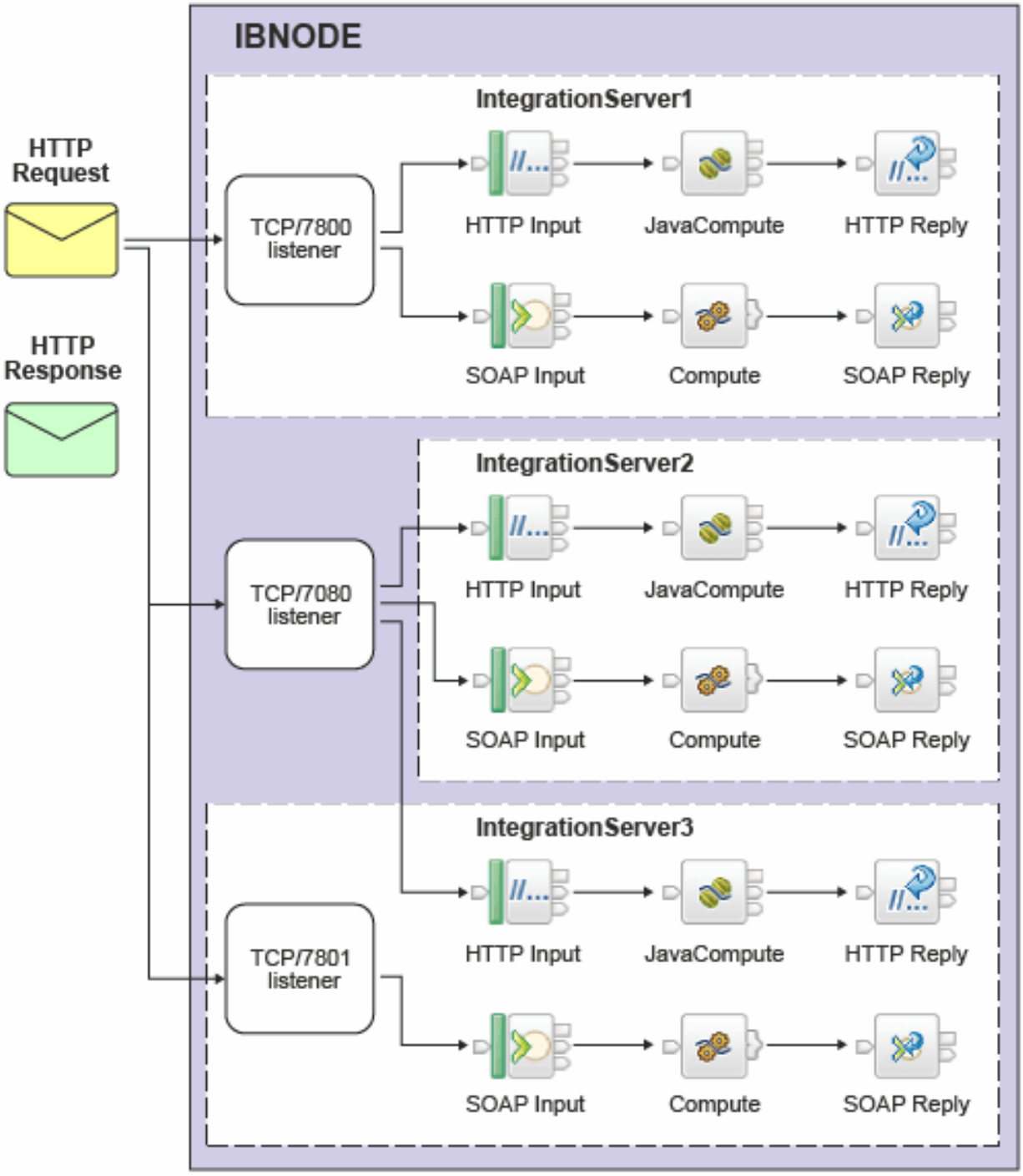
  This option is set as the default configuration for HTTP nodes for both existing and new integration nodes.
- You can use the listener that is embedded within each integration server, which also has two ports for HTTP and HTTPS messages.

  This option is set as the default configuration for SOAP nodes for both existing and new integration nodes.
- You can use a mixture of integration nodes and integration server listeners by keeping or making the integration node listener active, and configuring a subset of integration servers to use the embedded listener. Message flow nodes of the same type within one integration server must use the same type of listener. For example, you cannot have SOAP nodes that use the embedded listener and SOAP nodes that use the integration node listener within the same integration server.

The integration node listener requires access to system queues on the queue manager specified on the integration node, so you must install IBM MQ Server if you want to use an integration node listener. However, if you use HTTP nodes or SOAP nodes with the integration server embedded listener, IBM MQ is not required.

For more information about why you might choose each option, and how to configure them, see "HTTP listeners" on page 795.

The following diagram shows the use of both types of listener, configured on default ports, for HTTP



messages.

You must always use the correct reply node that matches your input node; you cannot combine an HTTPReply node with a SOAPInput, or a SOAPReply node with an HTTPInput node. An exception is generated when the reply is attempted.

You can include the reply node in the same message flow, or in a different message flow:

- If you have configured the HTTPInput node or SOAPInput node to use the integration server listener, you must deploy the second message flow to the same integration server.
- If you have configured the HTTPInput node to use the integration node listener, you can deploy the second message flow to any other integration server defined to the integration node. This is not the case for the SOAPInput node.

- You must pass the correct reply identifier from the input message flow to the reply node.

If you are using SOAP nodes and HTTP nodes in message flows on a single integration node, you can choose to handle HTTP messages by using either the integration node listener or embedded integration server listeners. If a listener in your configuration receives messages that both SOAPInput and HTTPInput nodes might get, you must carefully check the URL specifications in these nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

For more information about using the WebSphere Broker HTTP Transport, see the following topics:

- "Working with HTTP flows" on page 792
- "HTTP listeners" on page 795
- "HTTP message format" on page 790
- "HTTP headers" on page 791
- "Web services example messages" on page 804

For information about using HTTPS, see "Implementing SSL authentication" on page 2635.

You can also use the HTTP proxy servlet in an external Web servlet container to provide listener support for a larger number of concurrent HTTP sessions. For more information about the servlet and its uses, see "HTTP proxy servlet overview" on page 141.

Web services: when to use SOAP or HTTP nodes

HTTP and SOAP nodes can both be used to interact with web services. Typically you use SOAP nodes when working with SOAP-based web services.

For SOAP-based web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

Cases where it might be better to use HTTP nodes include:

- Message flows that interact with web services that use different standards, such as REST or XML-RPC.
- Message flows that never use WS-Addressing, WS-Security, SwA, or MTOM.

HTTP message format

An HTTP message contains components that are appropriate to its type.

The bit stream containing headers and body is parsed and represented within the message tree when an input request is received by an HTTPInput node, or when a response from a web service is received by the HTTPRequest node. A bit stream is created by parsers from the appropriate parts of the message tree when a reply is sent to the client by the HTTPReply node, or when a message is sent by the HTTPRequest node. For further details about these actions, see the individual node descriptions.

HTTP headers

When an HTTPInput or HTTPRequest node receives a message, it parses the HTTP headers to create elements in the message tree. When an HTTPReply or HTTPRequest node sends a message, it parses the HTTP headers from the message tree into a bit stream.

The HTTP headers in a message depend on the type of message that is processed. There are four message types recognized in a message flow, and a parser is associated with each of these.

1. Input. An input message is received by the HTTPInput node from a client. The HTTP headers in the input message (data up to and including the CRLFCRLF) are parsed by the HTTPInput parser and are included in the message tree under the correlation name HTTPInput. The headers shown in the following table are expected in an input message; others might also be present.

Header	Content	Example
Host	The host name to which the client issued the message.	localhost
Content-Length	The length of the body of the input message in decimal (that follows the CRLFCRLF after the last header).	520
Content-Type	The type of the body data.	text/xml; charset=utf-8
SOAPAction		"" (empty string)

   The headers in the following table might also be automatically generated by the HTTPInput node depending on the request.

Header	Content	Example
X-Original-HTTP-Command	An expanded version of the original inbound request	POST http://localhost:7800/Wss001/services/Wss001 HTTP/1.1
X-Remote-Addr	The IP address of the client (or proxy if the client is connecting through a proxy)	127.0.0.1
X-Remote-Host	The host name or address of the client (or proxy if the client is connecting through a proxy)	localhost
X-Server-Name	The integration node's machine name	localhost
X-Server-Port	The integration node's port	7800
X-Query-String	The query string if present in the inbound URL (optional)	a=b&x=y
X-Scheme	The scheme through which the client is connected, either http or https	http

2. Reply. A reply message is sent by the HTTPReply node to the client that sent the corresponding input message. The headers in the reply message are created in the message tree under the correlation name HTTPReply, which is also the name of the parser used to parse that part of the message tree to a bit stream. You can create your own HTTPReply header in a Compute node, or you can configure

the HTTPReply node to create it by using default values, or values taken from the HTTPReply or HTTPResponse trees in the input message, or both.

You can set the HTTPReply Status Code in the local environment; for more information, see the instructions for setting the HTTP Status Code for a reply in “Working with HTTP flows” on page 792.

If the HTTPReply node creates a default HTTPReply header, it contains the headers and values shown in the following table.

Header	Value
Content-Length (if present in the input message)	The calculated length of the reply message body in decimal.
Content-Type	text/xml; charset=*ccsid of the message body*

3. Request. A request message is sent by the HTTPRequest node. The HTTP headers in this message must be created in the message tree under the correlation name HTTPRequest, and are parsed by the HTTPRequest parser when the message tree is parsed to a bit stream. You can create your own HTTPRequest header in a Compute node, or you can configure the HTTPRequest node to create it using default values, or values taken from the HTTPInput or HTTPRequest trees in the input message, or both. If the HTTPRequest node creates a default HTTPRequest header, it contains the headers and values shown in the following table.

Header	Value
Host	Value set in the *`Default Web Service URL`* property.
Content-Length	The calculated length of the request message body in decimal.
Content-Type	text/xml; charset=*ccsid of the message body*
SOAPAction	"" (empty string)
Content-Encoding	"gzip" or "deflate" if the `Use compression` property is set to `gzip`, `zlib` (`deflate`), or `deflate`.
Accept-Encoding	"gzip, deflate" if the `Accept compressed responses by default` property is selected.

4. Response. A response message is received by the HTTPRequest node from the application to which the corresponding request message was sent. The HTTP headers in the response message (data up to and including the CRLFCRLF) are parsed by the HTTPResponse parser and are included in the message tree under the correlation name HTTPResponse. The header shown in the following table is expected in a response message (though not required); others might also be present.

Header	Content	Example
Content-Length	The length of the response message body in decimal.	1585

“Web services example messages” on page 804 provides example messages that include these headers.

Working with HTTP flows

Read this information if you are using HTTP message flows to interact with HTTP applications, including web services and RESTful services.

About this task

- Using secure connections with HTTPS
- Using HTTP asynchronous request-response

- Setting the HTTP Status Code for a reply
- Using LocalEnvironment.Destination.HTTP.RequestIdentifier
- Setting the HTTPRequest node URL dynamically
- Setting `Generate default HTTP headers from reply or response` for the HTTPReply node
- Setting `Generate default HTTP headers from input` for the HTTPRequest node

You must decide which listener you want the HTTP nodes to use:

- The integration node listener (the httplistener component) has an HTTPConnector for handling HTTP messages, and an HTTPSConnector for handling HTTPS (HTTP over SSL) messages. Each connector has its own assigned port.

  The integration node listener requires access to system queues on the queue manager specified on the integration node, so you must install IBM MQ Server if you want to use an integration node listener.
- An embedded listener is part of each integration server. The embedded listener has an HTTPConnector and an HTTPSConnector, and each connector has its own assigned port.

  You can configure one or more integration servers so that all HTTPInput and HTTPReply that you deploy to that integration server use the embedded listener.

  The integration server embedded listener does not require access to IBM MQ.

For a discussion of the advantages of each listener, see "HTTP listeners" on page 795.

You can use `ProxyConnectHeaders` only with HTTPS (SSL) connections; these headers do not work with HTTP connections.

Using secure connections with HTTPS

If you are using an HTTPS connection when the server certificate is replaced, you can use the App Connect Enterprise administration REST API to reload the server certificate from the keystore and apply it dynamically to the running integration server. This means that if an HTTPS connection is in use when a security certificate is replaced, you can apply the new certificate without having to restart the integration server. For more information, see "Updating the HTTPS Connector resource manager by using the administration REST API" on page 440.

For information about setting up HTTPS connections, see "Implementing SSL authentication" on page 2635.

Using HTTP asynchronous request-response

To make an HTTP request and receive a response asynchronously, use the HTTPAsyncRequest and HTTPAsyncResponse nodes in your message flow. The HTTPAsyncRequest node processes the next message without waiting for the response, which is received by the HTTPAsyncResponse node on a different thread and in a new transaction. This means that many more requests can be processed using fewer threads.

For more information, see "Using HTTP asynchronous request-response" on page 803.

Setting the HTTP Status Code for a reply

The default HTTP Status Code is 200, which indicates success. If you want a different status code to be returned, take one of the following actions:

- Set your status code in the field Destination.HTTP.ReplyStatusCode in the local environment tree (correlation name OutputLocalEnvironment). This field overrides any status code that is set in an HTTPResponseHeader header. This action is the preferred option, because it provides the greatest flexibility.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPReplyHeader header.
- Set your status code in the field X-Original-HTTP-Status-Code in the HTTPResponseHeader header. This option is useful if you include an HTTPRequest node before the HTTPReply node in your flow, because the HTTPResponseHeader header is created for you. In this scenario, an HTTPResponseHeader header has been created in the logical tree, representing the HTTP headers in the response from another web service. If you have selected the `Generate default HTTP`

`headers from reply or response` property on the HTTPReply node, values for the response header are set as default values when the reply message is created.

Using LocalEnvironment.Destination.HTTP.RequestIdentifier

When the HTTPInput node receives an input request message, it sets the local environment field Destination.HTTP.RequestIdentifier to a unique value that identifies the web service client that sent the request. You can refer to this value, and you can save it to another location if appropriate.

For example, if you design a pair of message flows that interact with an existing IBM MQ application you can save the identifier value in the request flow, and restore it in the reply flow, to ensure that the correct client receives the reply. If you use this technique, you must not change the data, and you must retain the data as a BLOB.

The HTTPReply node extracts the identifier value from the local environment tree and sets up the reply so that it is sent to the specific client. However, if you are using an HTTPReply node in a flow that does not have an HTTPInput node, and this field has been deleted or set incorrectly, message BIP3143S is issued.

If you design a message flow that includes both an HTTPInput and an HTTPReply node, the identifier value is set into the local environment by the HTTPInput node, but the HTTPReply node does not use it. Therefore, if your message flow includes both HTTP nodes and a Compute node in the same flow, you do not have to include the local environment tree when you specify which components of the message tree are copied from input message to output message by the Compute node (the `Compute mode` property).

Setting the HTTP request URL dynamically

You can set the property `Default web service URL` on the HTTPRequest or HTTPAsyncRequest node to determine the destination URL for a web service request. You can configure a Compute node before the HTTPRequest or HTTPAsyncRequest node within the message flow to override the value set in the property. Code ESQL that stores a URL string in LocalEnvironment.Destination.HTTP.RequestURL; the node retrieves and uses the URL string in place of the node property value.

Although you can also set the request URL in the special header X-Original-HTTP-URL in the HTTPRequestHeader header section of the request message (which overrides all other settings) in a Compute node, use the local environment tree content for this purpose for greater flexibility.

Setting `Generate default HTTP headers from reply or response` for the HTTPReply node

If you select `Generate default HTTP headers from reply or response` in the HTTPReply node properties, the node includes a minimum set of headers in the response that is sent to the web service client.

To set headers explicitly, create them in an HTTPReplyHeader header. For example, a Compute node propagates a message in the XMLNSC domain and modifies the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPReplyHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNSC = InputRoot.XMLNSC;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the reply is built by selecting the headers using the algorithm defined in the following steps:

1. Select one or more headers in an HTTPReplyHeader header.
2. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.
3. Select one or more headers in an HTTPResponseHeader header (an HTTPResponseHeader header is propagated on return from an HTTPRequest node).
4. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*.

5. Create or overwrite the Content-Length header.

Attention: The HTTPReply node always rewrites the Content-Length header, even if you have cleared `Generate default HTTP headers from reply or response`. This action ensures that the content is correct.

If an HTTPReplyHeader header section existed in the message received by the HTTPReply node, and the Output terminal of the HTTPReply node is connected, the HTTPReplyHeader header section is updated with all changed or added values.

Setting `Generate default HTTP headers from input` for the HTTPRequest or HTTPAsyncRequest node

If you select `Generate default HTTP headers from input` in the HTTPRequest or HTTPAsyncRequest node properties, the node includes a minimum set of headers in the request that is sent to the server.

To explicitly set headers, create them in an HTTPRequestHeader header. For example, a Compute node propagating a message in the XMLNSC domain can modify the Content-Type as follows:

```
CALL CopyMessageHeaders();
SET OutputRoot.HTTPRequestHeader."Content-Type" = 'text/xml';
SET OutputRoot.XMLNSC = InputRoot.XMLNSC;
```

Do not use the ContentType property to set the Content-Type unless you are working in the MIME domain. The ContentType property is intended to set the value of Content-Type used in MIME.

The full set of HTTP headers used in the request is built by selecting the headers using the algorithm defined in the following steps:

1. Set the Host header, based on either the request URL or the incoming HTTPRequestHeader header section of the message.
2. Select one or more headers in an HTTPRequestHeader header.
3. If no Content-Type header is yet defined, create one by using a non-empty value in the ContentType property.
4. Select one or more headers in an HTTPInputHeader header (an HTTPInputHeader header is created automatically by an HTTPInput node).
5. If no Content-Type header is yet defined, create one with the default value `text/xml; charset=`*`ccsid of the message body`*
6. If no SOAPAction header is yet defined, create one with the default value `''`.
7. Create or overwrite the Content-Length header.

Attention: The HTTPRequest or HTTPAsyncRequest node always rewrites the Content-Length header, even if you have cleared `Generate default HTTP headers from input or request`. This action ensures that the content is correct.

If an HTTPRequestHeader header exists in the received message, the HTTPRequestHeader header is updated with all changed or added values.

HTTP listeners

You can choose between integration node listeners and integration server (embedded) listeners to manage HTTP messages in your HTTP or SOAP flows. Learn about the two types of listener, how ports are assigned to them, and how you can switch from one to the other for individual integration servers.

Your choice of listener affects message flows that handle inbound web service requests and synchronous replies by using the following nodes:

- SOAPInput with SOAPReply
- HTTPInput with HTTPReply

Message flows that do not handle inbound requests but that instigate outbound requests or asynchronous responses by using the following nodes are not affected:

- SOAPRequest
- SOAPAsyncRequest
- SOAPAsyncResponse
- HTTPRequest
- HTTPAsyncRequest
- HTTPAsyncResponse

The choice of listener also affects integration services and REST APIs. Integration services use a SOAPInput node, so they use the listener that is specified for the SOAPInput node. REST APIs use an HTTPInput node, so they use the listener that is specified for the HTTPInput node.

The integration node listener requires access to system queues on the queue manager specified on the integration node, so you must install IBM MQ Server if you want to use an integration node listener. However, if you use HTTP nodes or SOAP nodes with the integration server embedded listener, IBM MQ is not required.

- "Integration server (embedded) listeners" on page 796
- "Integration node listeners" on page 797
- "Using both integration node and embedded listeners" on page 798

Integration server (embedded) listeners

Each integration server can run an embedded listener. You can configure the listener for HTTP or HTTPS configurations by setting values in the integration server's `server.conf.yaml` file:

```
HTTPConnector:
    #ListenerPort: 0                       # Set non-zero to set a specific port, defaults to
 7800.
    #ListenerAddress: '0.0.0.0'            # Set the IP address for the listener to listen on.
 Default is to listen on all addresses.
    #AutoRespondToHTTPHEADRequests: false   # Automatically respond to HTTP HEAD requests
 without invoking the message flow. Set to true or false, default is false.
    #ServerName: ''                        # Set the value to be returned in the 'Server' HTTP
 header.
    #CORSEnabled: false                    # Set the value to true to make the listener respond
 to valid HTTP CORS requests.
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'
HTTPSConnector:
    HTTPSConnector:
    #ListenerPort: 0                       # Set non-zero to set a specific port, defaults to
 7843.
    #ListenerAddress: '0.0.0.0'            # Set the IP address for the listener to listen on.
 Default is to listen on all addresses
    #ReqClientAuth: true
    #KeyAlias: ''
    #KeyPassword: 'myPassword'             # Set the password or alias to the password of the
 key
    #KeystoreFile: '/path/to/keystore.jks'
    #KeystorePassword: 'myPassword'        # Set the password or alias to the password of the
 keystore
    #KeystoreType: 'JKS'                   # Set the keystore type, can be 'JKS' or 'p12'.
 Default is JKS.
    #TruststoreFile: /path/tp/truststore.jks
    #TruststorePassword: 'myPassword'      # Set the password or alias to the password of the
 keystore
    #TruststoreType: 'JKS'                 # Set the truststore type, can be 'JKS' or 'p12'.
 Default is JKS.
    #CORSEnabled: false                    # Set the value to true to make the listener respond
 to valid HTTP CORS requests
    #CORSAllowOrigins: '*'
    #CORSAllowCredentials: false
    #CORSExposeHeaders: 'Content-Type'
    #CORSMaxAge: -1
    #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
    #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'
```

```
    #EnableTLSTrace: false                  # Enables tracing of TLS handshake messages to the
 console
```

For more information about these parameters, see Integration server HTTP listener parameters (SOAP and HTTP nodes).

The following examples assume that an integration server is associated with an integration node.

Run the following command to display the HTTPConnector properties for an integration server called *default* on integration node *integrationNodeName*:

```
mqsireportproperties integrationNodeName -e default -o HTTPConnector -r
```

Run the following command to display the HTTPSConnector properties for an integration server called *default* on integration node *integrationNodeName*:

```
mqsireportproperties integrationNodeName -e default -o HTTPSConnector -r
```

By default, SOAPInput, SOAPReply, SOAPAsyncResponse, HTTPInput, HTTPReply, and HTTPAsyncResponse nodes use the integration server listener. They also use the integration server listener if the integration node listener is not available, even if you do not explicitly reconfigure them to use the integration server listener. If integration server *default* on integration node *integrationNodeName* is using the embedded listener for SOAP nodes or HTTP nodes, the following commands return a value of `true`:

```
mqsireportproperties integrationNodeName -e default -o ExecutionGroup -n
 soapNodesUseEmbeddedListener
```

```
mqsireportproperties integrationNodeName -e default -o ExecutionGroup -n
 httpNodesUseEmbeddedListener
```

If you configure any of the SOAP nodes or HTTP nodes to use the integration node listener, the previous commands return a value of `false`.

Integration node listeners

You can configure the integration node listener for HTTP and HTTPS configurations by setting values in the integration node's `node.conf.yaml` file:

```
HTTPConnector:
   ListenerPort:  7080
HTTPSConnector:
   ListenerPort:  7083
```

Run the following two commands to display the currently active settings for the integration node listener:

```
mqsireportproperties integrationNodeName -b NodeHttpListener -o HTTPConnector -r
```

```
mqsireportproperties integrationNodeName -b NodeHttpListener -o HTTPSConnector -r
```

Each connector has its own assigned port; default values are `7080` for HTTP and `7083` for HTTPS. You can change these port numbers by using the **mqsichangeproperties** command. For example, run the following command to change the port on which the integration node listener for integration node *integrationNodeName* listens for HTTP messages:

```
mqsichangeproperties integrationNodeName -b NodeHttpListener -o HTTPConnector
   -n ListenerPort -v 8085
```

Run the following command to change the port on which the integration node listener for integration node *integrationNodeName* listens for HTTPS messages:

```
mqsichangeproperties integrationNodeName -b NodeHttpListener -o HTTPSConnector
   -n ListenerPort -v 8086
```

You can configure one or more integration servers so that HTTP nodes that you deploy to those integration servers use the embedded listener, or so that SOAP nodes that you deploy to those integration servers use the integration node listener.

Using both integration node and embedded listeners

Because the option to use the embedded listener is at the integration server level, you can change your configuration such that some integration servers that are associated with an integration node use the integration node listener for HTTP nodes, SOAP nodes, or both, and other integration servers use the embedded listener for HTTP nodes, SOAP nodes, or both.

However, if you disable the integration node listener, the integration server listeners are used for all HTTP and SOAP nodes, even if you do not explicitly enable support for them.

The HTTPRequest node always communicates directly through the HTTP transport, and is therefore unaffected by your choice.

If you change the listener and port that are processing your HTTP or HTTPS messages, you must ensure that you also update your applications to use the updated configuration.

Switching from embedded listeners to an integration node listener
Configure your integration node and integration servers that are associated with an integration node to use the integration node listener for HTTP or SOAP nodes in one or more integration servers.

Before you begin

The integration node listener requires access to system queues on the queue manager specified on the integration node, so you must install

IBM MQ Server and ensure that a queue manager is specified.

About this task

You can change the configuration for integration servers that are associated with an integration node so that HTTP or SOAP nodes use the integration node listener. HTTP nodes and SOAP nodes use the integration server (embedded) listener by default, but you might want to switch to using the integration node listener.

The commands that are shown in the examples here are split across multiple lines for ease of reading. When you enter the command, you must use a single line.

Procedure

1. Check that the integration node is running.
2. If you disabled the integration node listener, run the **mqsichangeproperties** command to restart it.
   For example:

   ```
   mqsichangeproperties integrationNodeName  -b NodeHttpListener -o HTTPListener
    -n startListener -v true
   ```

3. To switch to using the integration node listener for a specific integration server that is associated with the integration node, use the **mqsichangeproperties** command to change the integration server configuration.
   - The integration node must be stopped for the switch to work. If the node is running, stop it by using the following command

     mqsistop *integrationNodeName*
   - To use the **mqsichangeproperties** command, adapt one of the following examples:

     ```
     mqsichangeproperties integrationNodeName -f -e server_name -o ExecutionGroup
     ```

```
 -n httpNodesUseEmbeddedListener -v false
```

```
mqsichangeproperties integrationNodeName -f -e server_name -o ExecutionGroup
 -n soapNodesUseEmbeddedListener -v false
```

4. Optional: If you want to use HTTP applications on multiple integration servers, you need to change the HTTP/HTTPS ports to be unique for each integration server. By default the HTTP port is set to 7800 and the HTTPS port is set to 7843. Configure your integration servers by setting properties in the `server.conf.yaml` file, as described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172
5. Restart the integration node to ensure that your changes take effect.

Example

For more information about this command, and examples of changing other properties that are associated with an integration node or integration server, see the description of the **mqsichangeproperties** command.

Switching from an integration node listener to embedded listeners
Configure your integration nodes and associated integration servers so that some or all of the HTTP or SOAP nodes use an integration server (embedded) listener. SOAP nodes use an embedded listener by default. You might have changed them to use the integration node listener, but now want to return to using embedded listeners.

About this task

You can change the configuration for one or more integration servers that are associated with an integration node so that HTTP or SOAP nodes that are deployed in these integration servers use the embedded listener.

Procedure

1. Check that the integration node is running.
2. If you want all HTTP and SOAP nodes in all integration servers on the integration node to use the embedded listener, you can change the integration node configuration to disable the integration node listener.

   Run the **mqsichangeproperties** command to change the integration node configuration. Do not run this command if you want to keep the integration node listener active for at least one of your associated integration servers.

   ```
   mqsichangeproperties INODE  -b NodeHttpListener -o HTTPListener
    -n startListener -v false
   ```

   All integration servers on the specified integration node detect this change of status, and use the embedded listener when they are restarted, regardless of their own specific configuration. Therefore, you can switch to using embedded listeners for all associated integration servers by running this single command.

   If you disable the integration node listener in this way, you can configure an associated integration server to use the same port or ports that the integration node listener was using for HTTP, HTTPS, or both. Reusing the port numbers means that you do not have to change your client applications to send messages to a different port number.

3. To switch to using the embedded listener for a specific integration server on an integration node, use the **mqsichangeproperties** command to change the integration server configuration.
   - The integration node must be stopped for the switch to work. If the node is running, stop it by using the following command

     mqsistop *integrationNodeName*
   - To use the **mqsichangeproperties** command, adapt one of the following examples:

     ```
     mqsichangeproperties INODE -f -e exgroup1 -o ExecutionGroup
      -n soapNodesUseEmbeddedListener -v true
     ```

     ```
     mqsichangeproperties INODE -f -e exgroup1 -o ExecutionGroup
      -n httpNodesUseEmbeddedListener -v true
     ```

     INODE is the name of your integration node; *exgroup1* is the name of your integration server.
4. Restart the integration node to implement your changes.

Example

For more information about this command, and examples of changing other properties associated with integration nodes or integration servers, see the description of the **mqsichangeproperties** command.

Configuring message flows to process timeouts
Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Before you begin

- Read the message flows overview.
- Read about the options that you have for processing web service messages, and learn more about SOAP and HTTP.

About this task

You can configure message flows that start with an HTTPInput or SOAPInput node by connecting the HTTP Timeout terminal to further nodes for processing timeouts:

- On HTTPInput nodes, messages are propagated through this terminal only when you have configured your integration servers such that the HTTP nodes are using the embedded integration server listener.
- On SOAPInput nodes, messages are propagated through this terminal only when you are using an HTTP binding, and you have configured your integration servers such that the SOAP nodes are using the embedded integration server listener.

If these conditions are not met when you deploy the BAR file for the message flow that includes one of these nodes, a warning is generated, and the path of the message flow that you have connected to the HTTP Timeout terminal is ignored. No further warnings are generated until the next restart.

To set a static timeout value in an input node:

1. Create a message flow, or open an existing flow.
2. In the Message Flow editor, select the input node for this message flow. The node properties are displayed in the Properties view (below the editor pane).
3. Set an appropriate time for the timeout interval in the property `Maximum client wait time`. The default interval is 180 seconds.

   If this time expires, and you have not connected one or more nodes to the HTTP Timeout terminal, the listener that received the client request message responds with a SOAP Fault message indicating that a timeout has occurred.

4. If you want to provide customized timeout processing, connect one or more nodes to the HTTP Timeout terminal. You must include in this sequence the reply node that matches the input node. Therefore, if your message flow starts with an HTTPInput node, you must include an HTTPReply; if your message flow starts with a SOAPInput node, you must include a SOAPReply node.

To set a dynamic timeout value in an input node:

- Override the timeout value set on the input node by using the Java plug-in API to update it in a JavaCompute node using the `MbUtilities.changeIdentifierTimeout()` method. The following code shows an example of the `changeIdentifierTimeout` method:

```
MbMessage localEnv = assembly.getLocalEnvironment();
MbElement rootElem = localEnv.getRootElement();
MbElement repIdElement = rootElem.getFirstElementByPath(
   "/Destination/SOAP/Reply/ReplyIdentifier");
Object repId = repIdElement.getValue();
boolean success = changeIdentifierTimeout((byte[])repId, timeout);
```

- Override the timeout value set on the input node by using the CHANGEIDENTIFIERTIMEOUT function.

You can derive the value that you use to replace the existing value by several means; for example:

- Read a record from a database.
- Use a value from a field within the message body.

By propagating from the HTTP Timeout terminal you can then change the contents of the responses that your message flow sends to the client. The processing on the sequence of nodes that you connect to the HTTP Timeout terminal is also subject to a further timeout, so that the client always gets a response within a known timeout interval.

When a message is propagated from the HTTP Timeout terminal the message tree contains the input headers of the original input message and a message body that is the fault timeout message. The original message body along with other information relating to the timeout can be accessed in the LocalEnvironment message tree. For example, the following record can be found in the LocalEnvironment:

```
(0x01000000:Name):HTTP = (
  (0x01000000:Name):Input = (
    (0x01000000:Name):Timeout = (
      (0x03000000:NameValue):OriginalClientLastWaitTime = 10 (INTEGER)
      (0x03000000:NameValue):OriginalClientWaitTime     = 15 (INTEGER)
      (0x03000000:NameValue):OriginalMessageMadeTheFlow = TRUE (BOOLEAN)
      (0x03000000:NameValue):OriginalRequestIdentifier  =
         X'485454500000000000000000000000c00c000000000000' (BLOB)
      (0x03000000:NameValue):OriginalInboundMessage     = X'3c3e' (BLOB)
    )
  )
)
```

For SOAPInput nodes the SOAPReply node connected on the HTTP Timeout terminal path must send a SOAP fault response message and the reply status code of 500 cannot be altered. For HTTPInput nodes any response message can be sent from the HTTP Timeout terminal and the reply status code can be changed by updating the `LocalEnvironment.Destination.HTTP.ReplyStatusCode` message tree field.

Using compression with HTTP and SOAP nodes
You can configure HTTP and SOAP nodes to use HTTP compression and decompression when sending and receiving messages.

The nodes can be configured to compress request messages, accept and decompress responses, and decompress input messages.

Background information

The HTTP transport supports the sending of compressed data. Specific HTTP header fields are used to indicate that the data is compressed, and which compression technique was used for the compression. When an HTTP client makes an HTTP request, it can specify that it accepts a compressed response, and

also which types of compressed data it accepts. The three compression techniques supported by the HTTP transport are GZIP, deflate, and compress.

The Accept-Encoding field is used in an HTTP request to indicate which encodings are accepted in a response message to the request. This field is used to specify the values of `gzip`, `deflate`, `compress` and `identity`, which are not case-sensitive. A value of `identity` indicates that the data must not be compressed. A wildcard of `*` indicates that any encoding is accepted. The Accept-Encoding field can also be empty, indicating that no encoding is accepted.

The Content-Encoding header field indicates the encoding that is applied to the data and it can contain the tokens of `gzip`, `deflate`, and `compress`, but not `identity`. When data is compressed for an HTTP message, the Content-Encoding field contains the name of the compression technique that was used, enabling the recipient to identify the compression scheme and correctly decompress the message data.

The TE header field is used in a request header to indicate which extension transfer encodings it will accept in the response. This is similar to the Accept-Encoding field.

The Transfer-Encoding field is similar to the Content-Encoding field, except that it is a property of the message and not of the entity. The Transfer-Encoding field is primarily used to indicate that a response message is chunked. It is possible to receive a Transfer-Encoding header that indicates other transfer encodings such as "chunked, gzip". In this case the gzip applies to the transfer of the data, not the actual data itself.

HTTP compression in IBM App Connect Enterprise

IBM App Connect Enterprise supports compression and decompression with the HTTP and SOAP nodes using the Accept-Encoding and Content-Encoding fields as follows:

- In a request node, such as an HTTPRequest or SOAPRequest node, if you select the property to allow compressed responses, the integration node sets the Accept-Encoding field on the outbound request.
- When compressing a request, the request nodes set the Content-Encoding header field to indicate that the data is compressed. When receiving a response, the request nodes check the Content-Encoding field to determine if the response is compressed. Multiple values in the header field indicate that the data has been compressed more than once using multiple compression functions.
- When receiving a request, the input nodes check the contents of the Content-Encoding field to determine if the contents of the message are compressed. In response and input nodes, a Content-Encoding value of `x-gzip` is treated as `gzip`.
- IBM App Connect Enterprise supports only the `gzip` and `deflate` compression encodings; the `compress` encoding is not supported.
- Compression and decompression is handled only through the Accept-Encoding and Content-Encoding header fields; the TE and Transfer-Encoding header fields are not supported.

When compressing a request HTTP message, the node checks the Content-Encoding field to determine if the message data is already compressed. If the data is already compressed in the specified scheme then no further compression is needed. However, if the existing data is already compressed in an encoding that is not specified in the node properties, the node further compresses the compressed bit stream using the encoding specified in the node properties. The value of the Content-Encoding field is updated to indicate the additional encoding applied to the data. For example, a Content-Encoding value of `deflate,gzip` indicates that the message should be first decompressed using deflate and then further decompressed using gzip.

The HTTP and SOAP nodes do not support quality values in the Accept-Encoding field, which allow a user to specify a preferred weighting of compression types for responses. Any quality values in the Accept-Encoding field are ignored.

Using HTTP compression with the HTTP and SOAP nodes

The request nodes can request and process compressed responses. You can configure the request nodes to indicate that compression in responses is allowed, and the node automatically decompresses a

compressed response. The Accepts-Encoding header is set to indicate that GZIP and deflate compression techniques are accepted. If the Accept-Encoding header is already set, the node does not override it.

If the request or the AsyncResponse nodes receive a GZIP or deflate compressed response, it is decompressed and any header indication that the message is compressed is removed. If the nodes receive an invalid compressed response or an unrecognized compression function, an exception is raised indicating that the data could not be decompressed.

The request nodes can also send compressed requests. You can configure the node to specify which compression technique is used for the compressed requests that it sends. The value of the Content-Encoding header is set to indicate the compression that is used. You can override this value in the local environment for an individual message. If you override the local environment with a value that is not recognized by the node, the existing node value for Use compression is used.

The input nodes can decompress input data that is compressed using the GZIP and deflate compression scheme. If the input node receives a message that is not compressed validly, a fault message is returned to the client and the input message is not propagated to the message flow. This can occur if:

- The Content-Encoding header is set to an unrecognized compression function
- The message body is not compressed correctly using the named Content-Encoding value.

Using HTTP asynchronous request-response
You can make HTTP requests and receive a response asynchronously using the HTTPAsyncRequest and HTTPAsyncResponse nodes, or the SOAPAsyncRequest and SOAPAsyncResponse nodes, in your message flow.

The SOAPAsyncRequest node supports two methods of asynchronous requests when using HTTP transport:

1. Using WS-Addressing to direct the response to the paired SOAPAsyncResponse node. The SOAPAsyncRequest node waits for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. A new HTTP connection is made by the backend server to reply to the SOAPAsyncResponse node. This is the default behavior.
2. Using HTTP asynchronous request-response. When the SOAPAsyncRequest property **Use HTTP asynchronous request-response** is selected, the SOAPAsyncRequest node passes the HTTP socket to the paired SOAPAsyncResponse node to allow the backend server to reply using the same socket. When this option is selected, WS-Addressing headers are sent, but are not required to be understood by the backend server. The WS-Addressing headers are not marked as mustUnderstand, and the replyTo header is set to anonymous. The node therefore uses asynchronous HTTP socket handling instead of WS-Addressing to make HTTP requests and receive an asynchronous response.

The HTTPAsyncRequest node always uses asynchronous HTTP socket handling to make asynchronous requests and responses.

The asynchronous request nodes return control to the flow without waiting for a response. This action frees the request thread to handle further requests, while the response is handled by the paired response node on a different thread and in a new transaction. The response node can be in a separate message flow, but it must be in the same integration server as its paired request node.

The HTTP asynchronous request-response behavior is asynchronous only because IBM App Connect Enterprise treats the request and the response as such, enabling the message flow to retrieve the next message without waiting for the response from the asynchronous request. The backend server sees the request-response as a typical synchronous HTTP request.

Scenarios

You might want to use HTTP asynchronous request-response in the following situations:

- When interacting with a backend server that has a high latency, using synchronous HTTP requests might result in many outstanding requests which require large numbers of threads to make enough

simultaneous connections to the backend server. Using asynchronous HTTP socket handling instead allows the response to be decoupled from the request.

- Using HTTP asynchronous request-response in the SOAPAsyncRequest is an alternative to using WS-Addressing with a non-anonymous replyTo header.

For information about using HTTP asynchronous request-response with the SOAPAsyncRequest node, see "Choosing asynchronous behavior for the SOAPAsyncRequest node" on page 814.

Limitations

When using the HTTPAsyncRequest node, or the SOAPAsyncRequest node with HTTP asynchronous request-response, the following limitations apply:

- Redirection is not supported. Messages with a redirection status code (3xx) are treated as an error, and the response is routed to the Error terminal of the response node while following the properties specified on the Error tab.
- WS-RM is not compatible with using HTTP asynchronous request-response.

Web services example messages

Examples of complete HTTP messages show typical content in specific scenarios.

The following examples show complete HTTP messages. The first message is a request sent by an HTTPRequest node to a web service that provides a lookup service:

```
POST /greenpages/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 520
SOAP Action: ""
Cookie: JSESSIONID=0000B5OSLFIUDMQZFAUXKHD5ZDQ:-1

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema.xmlsoap.org.soap/envelope/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getUserByName xmlns:ns1="http://tempuri.org/imb.GreenPages"
                   SOAP-ENV:encodingStyle="http:/schemas.xmlsoap.org/soap/encoding/">
<nameField xsl:type="xsd:string">bloggs, joe</nameField>
</ns1:getUserByName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `Cookie` is an example of a value that can be retrieved from the HTTPRequest tree.

The second message is the corresponding web service response returned to the HTTPRequest node:

```
HTTP/1.0 200 OK
Server: WebSphere Application Server/4.0
Content-Type: text/xml; charset=utf-8
Content-Length: 1585
Content-Language: en
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schema.xmlsoap.org.soap/envelope/"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getUserByNameResponse xmlns:ns1="http://tempuri.org/imb.GreenPages"
                   SOAP-ENV:encodingStyle="http:/schemas.xmlsoap.org/soap/encoding/">
<return xmlns:ns2="http://www.greenpages.com/schemas/GreenPagesRemoteInterface"
        xsi:type="ns2:imb.UserRecord">
<fullName xsi:type="xsd:string">Joseph Bloggs</fullName>
<empNum xsi:type="xsd:int">65874</empNum>
<deskPhone xsi:type="xsd:string">(718)545-3623</deskPhone>
</return>
</ns1:getUserByNameResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information about HTTP return codes, see HTTP Response codes.

Processing web service messages

Use IBM App Connect Enterprise nodes and services to connect to other web services providers and consumers.

About this task

A web service is a software system designed to support interoperable computer-to-computer interaction over a network. It has an interface described by an XML-based specification; specifically, the Web Service Definition Language, or WSDL.

Web services fulfill a specific task or a set of tasks. A web service is described using a standard, formal XML notation, called its service description, that provides all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

The nature of the interface hides the implementation details of the service, so that it can be used independently of the hardware or software platform on which it is implemented. The interface is also independent of the programming language in which it is written. This interface handles web service-based applications as loosely coupled, component-oriented, cross-technology implementations. Web services can be used alone, or with other web services, to carry out a complex aggregation or a business transaction.

The IBM App Connect Enterprise environment provides two different development styles for developing web services:

- For an assisted development environment to provide web services, where you can focus on implementing service operations instead of directly handling the transport-level interaction, consider using Services. For more information, see “Developing integration solutions by using integration services” on page 1984.
- For more fine-grained control over developing a web service, consider creating your own message flows which use SOAPInput nodes directly. For more information, see “Message flows for web services” on page 820. You might want to do this in the following situations:
  - You are providing a web service over JMS.
  - You want to provide a gateway for multiple services.
  - You want more operational control over how web services are grouped.

To call a web service as part of your message flow, include SOAPRequest nodes in the message flow appropriately. For more information, see “Message flows for web services” on page 820.

For more information about how IBM App Connect Enterprise acts as a web service provider or consumer, and how it complies with external web service standards, see “IBM App Connect Enterprise and web services” on page 806.

When you have developed an IBM App Connect Enterprise solution, the IBM App Connect Enterprise administrator can apply policies to affect how messages are secured or when messages are retransmitted. The administrator also controls how the listener behaves, and can work with an external listener for HTTP traffic. For more information, see the following topics:

- “Web Services Reliable Messaging” on page 818
- “WS-Security” on page 2650
- “HTTP proxy servlet overview” on page 141

The following topics describe how to work with web services:

- “IBM App Connect Enterprise and web services” on page 806
- “What is SOAP?” on page 807
- “What is WSDL?” on page 815
- “What is SOAP MTOM?” on page 816

- "What is WS-Addressing?" on page 816
- "WebSphere Service Registry and Repository" on page 1005
- "Message flows for web services" on page 820

IBM App Connect Enterprise supplies a Java servlet that you can use in an external Web servlet container such as IBM WebSphere Application Server or Apache Tomcat, to receive HTTP requests from web services client applications. The HTTP proxy servlet is described in "HTTP proxy servlet overview" on page 141.

IBM App Connect Enterprise and web services

an IBM App Connect Enterprise application can participate in a web services environment as a service requester, as a service provider, or both.

The SOAP domain supports these formats:

- Common web services message formats SOAP 1.1, SOAP 1.2, SOAP with Attachments (SwA), and MTOM.
- Consistent SOAP logical tree format, which is independent of the exact message format.
- WS-Addressing, WS-Security, and WS-RM standards.

The following nodes are provided for use in the SOAP domain:

- node
- node
- node
- node
- node
- node
- node

Use the SOAP nodes and SOAP domain where possible; see "Web services: when to use SOAP or HTTP nodes" on page 790.

Web services support conforms to the following open standards:

- SOAP 1.1 and 1.2
- SOAP Messages with Attachments
- MTOM
- HTTP 1.1
- WSDL 1.1
- WS-Addressing (new SOAP domain only)
- WS-Security (new SOAP domain only)
- WS-RM
- WS-MakeConnection

WSDL is also validated against the WS-I Basic Profile Version 1.1. Conformance to the guidelines in this specification improves interoperability with other applications.

For more information about how an IBM App Connect Enterprise application can participate in a web services environment, see the IBM App Connect Enterprise web page on developerWorks®.

What is a web service?

A web service is defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine interaction over a network.

A web service fulfills a specific task or a set of tasks, and is described by a service description in a standard XML notation called Web Services Description Language (WSDL). The service description

provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location.

Other systems use SOAP messages to interact with the web service, typically by using HTTP with an XML serialization in conjunction with other Web-related standards.

The WSDL interface hides the details of how the service is implemented, and the service can be used independently of the hardware or software platform on which it is implemented, and independently of the programming language in which it is written.

Applications that are based on web services are loosely-coupled, component-oriented, cross-technology implementations. Web services can be used alone, or in conjunction with other web services to carry out a complex aggregation or a business transaction.

What is SOAP?
SOAP is an XML message format used in web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used. The use of SOAP in a specific web service is described by a WSDL definition.

There are two versions of SOAP in common use: SOAP 1.1 and SOAP 1.2. Both are supported in IBM App Connect Enterprise. SOAP is defined in the following documents issued by World Wide Web Consortium (W3C):

- Simple Object Access Protocol (SOAP) 1.1 (W3C note).
- SOAP Version 1.2 Part 0: Primer (W3C recommendation).
- SOAP Version 1.2 Part 1: Messaging Framework (W3C recommendation).
- SOAP Version 1.2 Part 2: Adjuncts (W3C recommendation).

Support for SOAP in IBM App Connect Enterprise includes:

- SOAP parser and domain. See "SOAP parser and domain" on page 526.
- SOAP nodes to send and receive messages in SOAP format.
- IBM supplied message definitions for SOAP 1.1 and SOAP 1.2. These message definitions support validation, ESQL content assist, and the creation of message maps for use with SOAP messages, in the SOAP and other XML domains. See Message Sets: IBM supplied messages that you can import.
- HTTP and JMS transport to send SOAP messages. See the W3C SOAP over JMS specification: http://www.w3.org/TR/soapjms/

WSDL validation in IBM App Connect Enterprise refers to the WS-I Basic Profile. For more information, see the WS-I, and in particular the WS-I Basic Profile document:

- http://www.ws-i.org/
- http://www.ws-i.org/deliverables

The structure of a SOAP message
A SOAP message is encoded as an XML document, consisting of an <Envelope> element, which contains an optional <Header> element, and a mandatory <Body> element. The <Fault> element, contained in <Body>, is used for reporting errors.

The SOAP envelope
<Envelope> is the root element in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element.

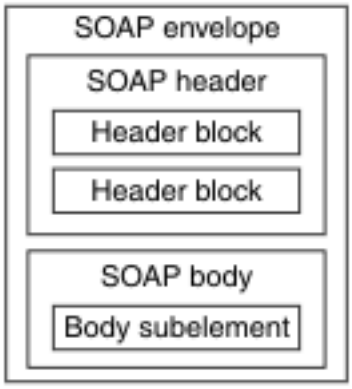
The SOAP header
<Header> is an optional subelement of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path; see "The SOAP header" on page 808.

The SOAP body
<Body> is a mandatory subelement of the SOAP envelope, which contains information intended for the ultimate recipient of the message; see "The SOAP body" on page 810.

The SOAP fault

<Fault> is a subelement of the SOAP body, which is used for reporting errors; see "The SOAP fault" on page 810.

XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure. The following diagram shows the structure of a SOAP message.



The following code is an example of a SOAP message that contains header blocks (the <m:reservation> and <n:passenger> elements) and a body (containing the <p:itinterary> element).

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
  env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

The SOAP header

The SOAP header (the <Header> element) is an optional sub-element of the SOAP envelope, and is used to pass application-related information that is processed by SOAP nodes along the message flow.

The immediate child elements of the header are called *header blocks*. A header block is an application-defined XML element, and represents a logical grouping of data which can be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver.

SOAP header blocks can be processed by SOAP intermediary nodes, and by the ultimate SOAP receiver node. However, in a real application, not every node processes every header block. Each node is typically designed to process particular header blocks, and each header block is processed by particular nodes.

The SOAP header enables you to add features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines some attributes that can be used to indicate what can deal with a feature and whether it is optional or mandatory. Such control information includes, for example, passing directives or contextual information related to the processing of the message. This control information enables a SOAP message to be extended in an application-specific manner.

Although the header blocks are application-defined, SOAP-defined attributes on the header blocks indicate how the header blocks must be processed by the SOAP nodes. SOAP-defined attributes include:

encodingStyle
Indicates the rules used to encode the parts of a SOAP message. SOAP defines a narrower set of rules for encoding data than the flexible encoding that XML enables.

actor (SOAP 1.1) or role (SOAP 1.2)
In SOAP 1.2, the role attribute specifies whether a particular node will operate on a message. If the role specified for the node matches the role attribute of the header block, the node processes the header. If the roles do not match, the node does not process the header block. In SOAP 1.1, the actor attribute performs the same function.

Roles can be defined by the application, and are designated by a URI. For example, `http://example.com/Log` might designate the role of a node which performs logging. Header blocks that are processed by this node specify `env:role="http://example.com/Log"` (where the namespace prefix `env` is associated with the SOAP namespace name of `http://www.w3.org/2003/05/soap-envelope`).

The SOAP 1.2 specification defines three standard roles in addition to those which are defined by the application:

`http://www.w3.org/2003/05/soap-envelope/none`
None of the SOAP nodes on the message path should process the header block directly. Header blocks with this role can be used to carry data that is required for processing of other SOAP header blocks.

`http://www.w3.org/2003/05/soap-envelope/next`
All SOAP nodes on the message path are expected to examine the header block (provided that the header has not been removed by a node earlier in the message path).

`http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`
Only the ultimate receiver node is expected to examine the header block.

mustUnderstand
This attribute is used to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application. If a SOAP node determines, by using the **role** or **actor** attribute, that it should process a header block, the action taken depends on the value of the **mustUnderstand** attribute.

- 1 (SOAP 1.1) or `true` (SOAP 1.2): The node must either process the header block in a manner consistent with its specification, or not at all (and throw a fault).
- `0` (SOAP 1.1) or `false` (SOAP 1.2): The node is not obliged to process the header block.

In effect, the **mustUnderstand** attribute indicates whether processing of the header block is mandatory or optional.

relay (SOAP 1.2 only)
When a SOAP intermediary node processes a header block, the SOAP intermediary node removes the header block from the SOAP message. By default, the SOAP intermediary node also removes all header blocks that it ignored (because the **mustUnderstand** attribute had a value of `false`). However, when the relay attribute is specified with a value of `true`, the SOAP intermediary node retains the unprocessed header block in the message.

The SOAP body
The SOAP body (the <Body> element) is a mandatory sub-element of the SOAP envelope, which contains information intended for the ultimate recipient of the message.

The body element and its associated child elements are used to exchange information between the initial SOAP sender and the ultimate SOAP receiver. SOAP defines one child element for the body: the <Fault> element, which is used for reporting errors. Other elements in the body are defined by the web service that uses them.

The SOAP fault
The SOAP fault (the <Fault> element) is a sub-element of the SOAP body, which is used for reporting errors.

If present, the SOAP fault element must appear as a body entry and must not appear more than once in a body element. The sub-elements of the SOAP fault element are different in SOAP 1.1 and SOAP 1.2.

SOAP 1.1

In SOAP 1.1, the SOAP fault contains the following sub-elements:

<faultcode>
The<faultcode> element is a mandatory element in the <Fault> element. It provides information about the fault in a form that can be processed by software. SOAP defines a small set of SOAP fault codes covering basic SOAP faults, and this set can be extended by applications.

<faultstring>
The <faultstring> element is a mandatory element in the <Fault> element. It provides information about the fault in a form intended for a human reader.

<faultactor>
The <faultactor> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <faultactor> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<detail>
The <detail> element carries application-specific error information related to the <Body> element. It must be present if the contents of the <Body> element were not successfully processed. The <detail> element must not be used to carry information about error information belonging to header entries. Detailed error information belonging to header entries must be carried in header entries.

SOAP 1.2

In SOAP 1.2, the SOAP fault contains the following sub-elements:

<Code>
The <Code> element is a mandatory element in the <Fault> element. It provides information about the fault in a form that can be processed by software. It contains a <Value> element and an optional <Subcode> element.

<Reason>
The <Reason> element is a mandatory element in the <Fault> element. It provides information about the fault in a form intended for a human reader. The <Reason> element contains one or more <Text> elements, each of which contains information about the fault in a different language.

<Node>
The <Node> element contains the URI of the SOAP node that generated the fault. A SOAP node that is not the ultimate SOAP receiver must include the <Node> element when it creates a fault; an ultimate SOAP receiver is not obliged to include this element, but might do so.

<Role>
The <Role> element contains a URI that identifies the role in which the node was operating at the point the fault occurred.

<Detail>

The <Detail> element is an optional element, which contains application-specific error information related to the SOAP fault codes describing the fault. The presence of the <Detail> element has no significance as to which parts of the faulty SOAP message were processed.

SOAP nodes

The SOAP nodes act as points in the flow where web service processing is configured and applied. Properties on the SOAP nodes control the processing carried out and can be configured by supplying a WSDL definition, or by manually configuring properties, or both.

SOAP nodes

- The SOAPInput and SOAPReply nodes are used in a message flow which implements a web service. These SOAP nodes are used to construct a message flow that implements a web service provider. The SOAPInput node listens for incoming web service requests, and the SOAPReply sends responses back to the client; see node and node.
- A client can send an HTTP GET request to the endpoint exposed by the flow, suffixed with a query string ?wsdl, and receive a response with the WSDL definition used to configure the flow. For a full description, see “Message flow configuration with a WSDL file” on page 839.
- The SOAPRequest node is used in a message flow to call a web service provider synchronously. Calling a web service synchronously means that the node sends a web service request and waits, blocking the message flow, for the associated web service response to be received before the message flow continues; see node.
- The SOAPAsyncRequest and SOAPAsyncResponse nodes are used to construct a message flow (or pair of flows) which calls a web service asynchronously. Calling a web service asynchronously means that the SOAPAsyncRequest node sends a web service request, but the request does not block the message flow by waiting for the associated web service response to be received because the web service response is received at the SOAPAsyncResponse node, which is in a separate flow. The Node Correlator identifies the logical pairing of the responses against the original requests. Multiple requests can, therefore, be handled in parallel; see node and node.
- You can work on the payload of the SOAP body using the SOAPExtract and SOAPEnvelope nodes. The SOAPExtract node can interoperate with the SOAP domain. The SOAP nodes do not require the SOAPEnvelope node, because they can directly handle non-SOAP messages, but the SOAPEnvelope node is still required for the HTTP nodes. See node and node.
- You can change the Operation mode of the SOAP nodes so that they act in gateway mode. In gateway mode, a WSDL is not required to configure the nodes since they handle generic request/response and one-way SOAP messages that are not tied to a specific WSDL. For more details, see “Gateway operation mode for SOAP nodes” on page 827.

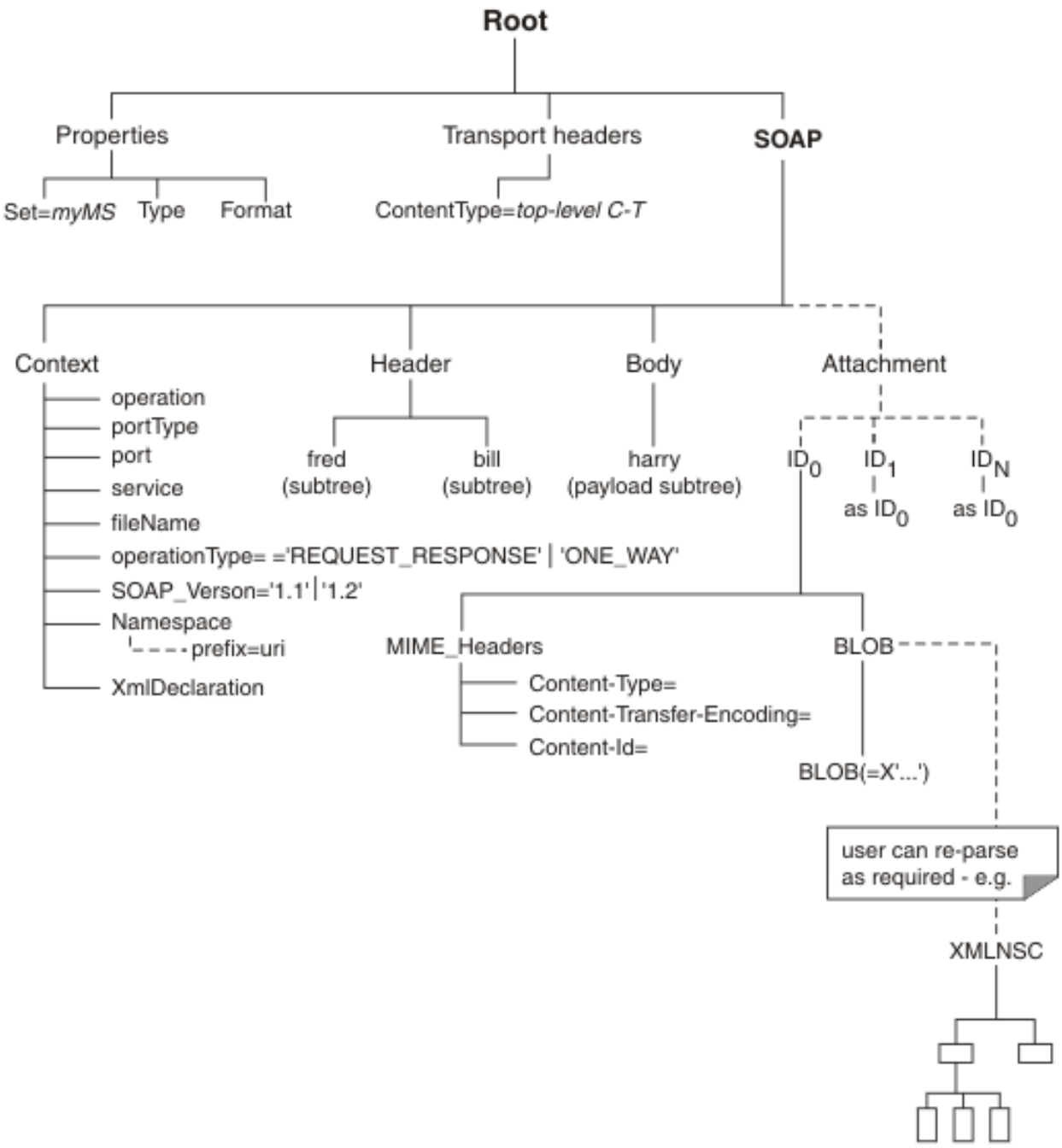
The W3C SOAP specification refers to "SOAP nodes" meaning a unit of application logic (see Web Services Glossary). Typically, references to "SOAP nodes" in the IBM App Connect Enterprise Information Center are referring to IBM App Connect Enterprise SOAP nodes.

If you are using SOAP nodes and HTTP nodes in message flows on a single integration node, you can choose to handle HTTP messages by using either the integration node listener or embedded integration server listeners. If a listener in your configuration receives messages that both SOAPInput and HTTPInput nodes might get, you must carefully check the URL specifications in these nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

SOAP tree overview

This tree format allows you to access the key parts of the SOAP message in a convenient way.

This is a diagrammatic representation of the SOAP domain tree:



The SOAP tree contains the following elements:

SOAP.Header
Contains the SOAP header blocks (children of `Envelope.Header`)

SOAP.Body
Contains the SOAP payload (children of `Envelope.Body`)

The content of the Body subtree depends on the WSDL style.

SOAP.Attachment
Contains attachments for an SwA message in their non encoded format.

Note that attachments for an MTOM message are represented inline as part of the SOAP content in a base 64 representation.

SOAP.Context
Contains the following information:

- Input; populated by the SOAPInput node:
  - `operation` - the WSDL operation name. In Gateway mode, the operation is assumed to be the name of the element that is the first child of the SOAP Body element, if present, otherwise it is the constant name 'ComIbmBrokerGenericGatewayOperation'.
  - `portType` - the WSDL port type name. In Gateway mode, this item is empty.
  - `port` - the WSDL port name (if known). In Gateway mode, this item is empty.
  - `service` - the WSDL service name (if known). In Gateway mode, the service has the constant name 'ComIbmBrokerGenericGatewayService'.
  - `fileName` - the original WSDL file name. In Gateway mode, this item is empty.
  - `operationType` - one of 'REQUEST_RESPONSE', 'ONE_WAY', 'SOLICIT_RESPONSE', 'NOTIFICATION'. In Gateway mode, without WSDL, this field contains 'GATEWAY'. This means 'REQUEST_RESPONSE' or 'GATEWAY_ONE_WAY', which means that the node has detected the operation type to be one-way.
  - `SOAP_Version` - one of '1.1' or '1.2'.
  - `Namespace` - Contains `nameValue` child elements; the `name` is the Namespace prefix, and the `value` is the Namespace URI as it appears in the bit stream.
  - `XmlDeclaration` - represents the standard XML declaration.
- Output; the following fields can be placed in `SOAP.Context` to provide override information when SOAPRequest or SOAPAsyncRequest nodes serialize a SOAP message:
  - `SOAP_Version` - one of '1.1' or '1.2'
  - `Namespace` - Contains `nameValue` child elements that define the namespace prefix (the `name`) to be used for a specified namespace URI (the `value`).

    An output message uses the namespace prefixes defined here to qualify any elements in the corresponding namespaces.

    If the `SOAP.Context` was originally created at an input node, it might already contain all the namespace prefix definitions that you need.

    If `SOAP.Context` does not exist, or the outgoing message uses additional namespaces, the SOAP parser generates any required namespace prefixes automatically.

    Alternatively, you can specify your own namespace prefix; the specific name of a namespace prefix does not usually affect the meaning of a message, with one important exception. If the message content contains a `qualified name`, the message must contain a matching namespace prefix definition.

    For example, if the output message is a SOAP Fault containing a `<faultcode>` element with the value `soapenv:Server`, a namespace prefix (which is case sensitive) for `soapenv` must be defined in the logical tree:

```
-- Build SOAP Fault message. Note that as well as defining the correct
-- namespace for the Fault element, it is also necessary to bind the
-- namespace prefix used in the faultcode element (this is set up under
-- SOAP.Context.Namespace)

-- Send back a new user defined SOAP 1.2 fault message
DECLARE soapenv NAMESPACE 'http://www.w3.org/2003/05/soap-envelope';
DECLARE xml     NAMESPACE 'http://www.w3.org/XML/1998/namespace';
DECLARE myNS    NAMESPACE 'http://myNS';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:myNS = myNS;

SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Value = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Code.soapenv:Subcode.soapenv:Value = 'my:subcode value';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text = 'my Reason string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Reason.soapenv:Text.(SOAP.Attribute)xml:lang = 'en';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Node = 'my Node string';
```

```
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Role = 'my Role string';
SET OutputRoot.SOAP.Body.soapenv:Fault.soapenv:Detail.my:Text = 'my detail string';
```

```
-- Send back a new user defined SOAP 1.1 fault message
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenv = soapenv;

SET OutputRoot.SOAP.Body.soapenv:Fault.faultcode = 'soapenv:Receiver';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultstring = 'my fault string';
SET OutputRoot.SOAP.Body.soapenv:Fault.faultactor = 'my fault actor';
SET OutputRoot.SOAP.Body.soapenv:Fault.detail.Text = 'my detail string';
```

Only `Namespace`, `SOAP_Version`, and `XmlDeclaration` influence the bit stream generated for a SOAP tree; the other fields are for information only.

To build a graphical data map for the SOAP domain, use the supplied IBM message for the SOAP domain tree as the input for the map, the output, or both. Use the Cast or Submap features of the Graphical Data Mapper to define the specific content of the SOAP body. See Using message maps for more information about mapping functions.

Web services: when to use SOAP or HTTP nodes
HTTP and SOAP nodes can both be used to interact with web services. Typically you use SOAP nodes when working with SOAP-based web services.

For SOAP-based web services, several advantages exist if you use the SOAP nodes and the SOAP message domain instead of the HTTP transport nodes and XMLNSC message domain.

- Support for WS-Addressing, WS-Security and SOAP headers.
- A common SOAP logical tree format, independent of the bitstream format.
- Runtime checking against WSDL.
- Automatic processing of SOAP with Attachments (SwA).
- Automatic processing of Message Transmission Optimization Mechanism (MTOM).

Although the HTTP nodes can process SwA messages, you must use the MIME message domain and design your flow to handle the attachments explicitly, and use custom logic to extract and parse the SOAP.

Cases where it might be better to use HTTP nodes include:

- Message flows that interact with web services that use different standards, such as REST or XML-RPC.
- Message flows that never use WS-Addressing, WS-Security, SwA, or MTOM.

Choosing asynchronous behavior for the SOAPAsyncRequest node
Choose between WS-Addressing and HTTP asynchronous request-response to make asynchronous requests using the SOAPAsyncRequest node with HTTP transport.

Before you begin

Read the concept information about “Using HTTP asynchronous request-response” on page 803.

About this task

The SOAPAsyncRequest node can use HTTP or JMS transport. It is linked as a pair with a SOAPAsyncResponse node using a unique identifier to correlate response messages with the original request.

The SOAPAsyncRequest node sends a web service request, but the node does not wait for the associated web service response to be received. This asynchronous functionality enables multiple outbound requests to be made almost in parallel because the outbound request is not blocked waiting for the response. The web service response is received by the SOAPAsyncResponse node, which can be in a separate message flow.

The SOAPAsyncRequest node supports two methods of asynchronous requests when using HTTP transport:

1. Using WS-Addressing to direct the response to the paired SOAPAsyncResponse node. The SOAPAsyncRequest node waits for the HTTP 202 acknowledgment before continuing with the message flow, and the SOAPAsyncRequest node blocks if the acknowledgment is not received. A new HTTP connection is made by the backend server to reply to the SOAPAsyncResponse node. This is the default behavior.
2. Using HTTP asynchronous request-response. When the SOAPAsyncRequest property **Use HTTP asynchronous request-response** is selected, the SOAPAsyncRequest node passes the HTTP socket to the paired SOAPAsyncResponse node to allow the backend server to reply using the same socket. When this option is selected, WS-Addressing headers are sent, but are not required to be understood by the backend server. The WS-Addressing headers are not marked as mustUnderstand, and the replyTo header is set to anonymous. The node therefore uses asynchronous HTTP socket handling instead of WS-Addressing to make HTTP requests and receive an asynchronous response.

Typically you should choose the HTTP asynchronous request-response method to make asynchronous requests using the SOAPAsyncRequest node with HTTP transport. To use this behavior, ensure that the SOAPAsyncRequest node property **Use HTTP asynchronous request-response** is selected.

However, you might want to use WS-Addressing instead to make asynchronous requests using the SOAPAsyncRequest node in the following cases:

- If you want to use WS-Addressing to explicitly set the replyTo header for the response message
- If the backend server has a high latency, you might want to use WS-Addressing instead of HTTP asynchronous request-response so that the HTTP socket is not left open for a long time.

To use WS-Addressing instead of HTTP asynchronous request-response behavior when you are using HTTP transport, ensure that the SOAPAsyncRequest node property **Use HTTP asynchronous request-response** is cleared. This is the default behavior of the SOAPAsyncRequest node.

What is WSDL?
WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

IBM App Connect Enterprise supports WSDL 1.1, as defined in the following document issued by the World Wide Web Consortium (W3C): Web Services Description Language (WSDL) 1.1. IBM App Connect Enterprise support for WSDL also adheres to the Web Services Interoperability Organization (WS-I) Basic profile 1.1; see Web Services Interoperability Organization (WS-I).

A WSDL definition is divided into separate sections that specify the logical interface and the physical details of a web service. The physical details include both endpoint information, such as HTTP port number, and binding information, which specifies how the SOAP payload is represented and which transport is used.

Support for WSDL in IBM App Connect Enterprise includes:

- Import of WSDL to create message roots in an application or library; see "Importing from WSDL" on page 2305.
- Generation of WSDL from a message set; see Message Sets: WSDL generation.
- WSDL editor with text and graphical design views.
- Use of WSDL to configure nodes in the SOAP domain; for example, you can drag WSDL onto a node, and a client can request the WSDL that was used to configure a SOAPInput node. For more details, see "Message flow configuration with a WSDL file" on page 839

When you import or generate WSDL, the WSDL is validated against the WS-I Basic Profile. You must fix validation errors before the application, library, or message set can be deployed. Validation warnings do not prevent deployment, but can indicate potential interoperability problems. The validated WSDL becomes an integral part of the application, library, or message set.

The WSDL editor supports a graphical design view so that you can navigate from the WSDL to its associated message roots. The application or library contains all the message roots (or the message set contains all the message definitions) required by message flows that are working with the web service described by the WSDL. At development time, the message roots or definitions support ESQL

Content Assist and the creation of mappings. At run time, the deployed application, library, or message set supports schema validation in the SOAP, XMLNSC, and MRM domains. In the SOAP domain, runtime checks are also made against the WSDL itself, and WSDL information is included in the SOAP logical tree.

What is SOAP MTOM?
SOAP Message Transmission Optimization Mechanism (MTOM) is the use of MIME to optimize the bitstream transmission of SOAP messages that contain significantly large base64Binary elements.

The MTOM message format allows bitstream compression of binary data. Data that would otherwise have to be encoded in the SOAP message is instead transmitted as raw binary data in a separate MIME part. A large chunk of binary data takes up less space than its encoded representation, so MTOM can reduce transmission time, although it can increase processor usage. Candidate elements to be transmitted in this way are defined as base64Binary in the WSDL (XML Schema).

An MTOM message is identified by a Content-Type with a type of `application/xop+xml`.

The SOAP domain handles inbound MTOM messages automatically, and MTOM parts are reincorporated automatically into the SOAP Body.

The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes; for details, see "SOAP MTOM and the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes" on page 849.

For details of the external specification published by the World Wide Web Consortium (W3C), see SOAP MTOM.

What is WS-Addressing?
Web Services Addressing (WS-Addressing) is a World Wide Web Consortium (W3C) specification that aids interoperability between web services by defining a standard way to address web services and provide addressing information in messages.

Start here to find out how IBM App Connect Enterprise supports WS-Addressing.

The WS-Addressing specification introduces two primary concepts: endpoint references, and message addressing properties. This topic contains an overview of each concept. For further details, select the following links to access the WS-Addressing specifications:

- W3C WS-Addressing specifications
- W3C submission WS-Addressing specification

Endpoint references (EPRs)

EPRs provide a standard mechanism to encapsulate information about specific endpoints. EPRs can be propagated to other parties and then used to target the web service endpoint that they represent. The following table summarizes the information model for EPRs.

Abstract Property name	Property type	Multiplicity	Description
[address]	xs:anyURI	1..1	The absolute URI that specifies the address of the endpoint.
[reference parameters]*	xs:any	0..unbounded	Namespace qualified element information items that are required to interact with the endpoint.
[metadata]	xs:any	0..unbounded	Description of the behavior, policies and capabilities of the endpoint.

The following prefix and corresponding namespace is used in the previous table.

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema

The following XML fragment illustrates an endpoint reference. This element references the endpoint at the URI http://example.com/fabrikam/acct, has metadata specifying the interface to which the endpoint reference refers, and has application-defined reference parameters of the http://example.com/fabrikam namespace.

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                       xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                       xmlns:fabrikam="http://example.com/fabrikam"
                       xmlns:wsdli="http://www.w3.org/2005/08/wsdl-instance"
                       wsdli:wsdlLocation="http://example.com/fabrikam
                       http://example.com/fabrikam/fabrikam.wsdl">
    <wsa:Address>http://example.com/fabrikam/acct</wsa:Address>
    <wsa:Metadata>
        <wsaw:InterfaceName>fabrikam:Inventory</wsaw:InterfaceName>
    </wsa:Metadata>
    <wsa:ReferenceParameters>
        <fabrikam:CustomerKey>123456789</fabrikam:CustomerKey>
        <fabrikam:ShoppingCart>ABCDEFG</fabrikam:ShoppingCart>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Message addressing properties (MAPs)

MAPs are a set of well defined WS-Addressing properties that can be represented as elements in SOAP headers. MAPs can provide either a standard way of conveying information, such as the endpoint to which message replies should be directed, or information about the relationship that the message has with other messages. The MAPs that are defined by the WS-Addressing specification are summarized in the following table.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[action]	xs:anyURI	1..1	An absolute URI that uniquely identifies the semantics of the message. This proprety corresponds to the [address] property of the endpoint reference to which the message is addressed. This value is required.
[destination]	xs:anyURI	1..1	The absolute URI that specifies the address of the intended receiver of this message. This value is optional because, if not present, it defaults to the anonymous URI that is defined in the specification, indicating that the address is defined by the underpinning protocol.
[reference parameters]*	xs:any	0..unbounded	Correspond to the [reference parameters] property of the endpoint reference to which the message is addressed. This value is optional.
[source endpoint]	EndpointReference	0..1	A reference to the endpoint from which the message originated. This value is optional.
[reply endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of replies to this message. This value is optional.
[fault endpoint]	EndpointReference	0..1	An endpoint reference for the intended receiver of faults relating to this message. This value is optional.

Abstract WS-Addressing MAP name	MAP content type	Multiplicity	Description
[relationship]*	xs:anyURI plus optional attribute of type xs:anyURI	0..unbounded	A pair of values that indicate how this message relates to another message. The content of this element conveys the [message id] of the related message. An optional attribute conveys the relationship type. This value is optional.
[message id]	xs:anyURI		An absolute URI that uniquely identifies the message. This value is optional.

The abstract names in the previous tables are used to refer to the MAPs throughout this documentation.

The following example of a SOAP message contains WS-Addressing MAPs:

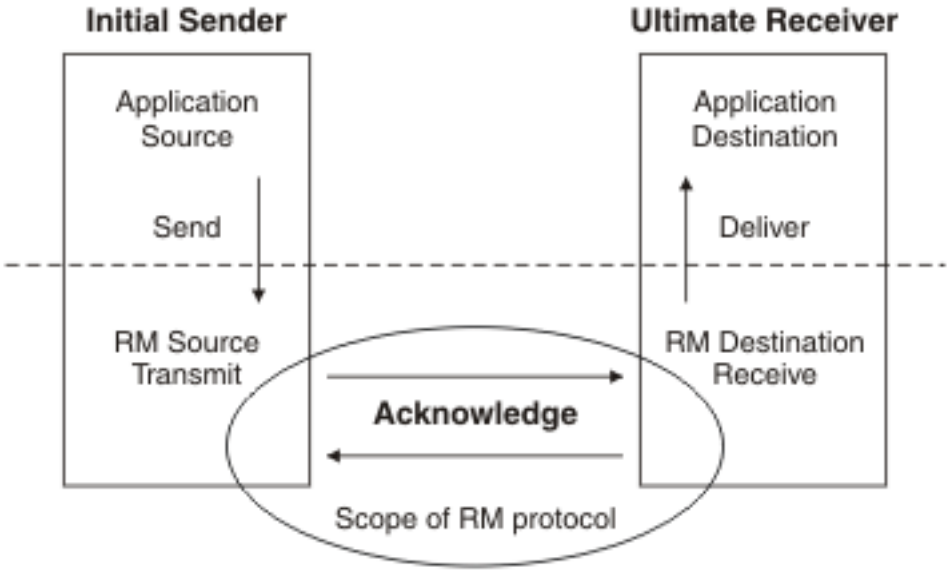
```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            xmlns:fabrikam="http://example.com/fabrikam">
    <S:Header>
        ...
     <wsa:To>http://example.com/fabrikam/acct</wsa:To>
              <wsa:ReplyTo>
        <wsa:Address> http://example.com/fabrikam/acct</wsa:address>
              </wsa:ReplyTo>
            <wsa:Action>...</wsa:Action>
              <fabrikam:CustomerKey wsa:IsReferenceParameter='true'>123456789
              </fabrikam:CustomerKey>
             <fabrikam:ShoppingCart wsa:IsReferenceParameter='true'>ABCDEFG
             </fabrikam:ShoppingCart>
              ...
    </S:Header>
        <S:Body>
          ...
        </S:Body>
</S:Envelope>
```

Web Services Reliable Messaging
IBM App Connect Enterprise supports WS-RM (Web Services Reliable Messaging), which allows two systems to reliably exchange messages with each other.

Web Services Reliable Messaging (WS-RM) is an OASIS standard that allows two systems to reliably exchange SOAP messages with each other. The purpose of WS-RM is to ensure delivery of messages in situations such as the destination endpoint being temporarily unavailable (for example, in the case of a server restart) or the message path crossing multiple transport connections, any of which might fail (for example, across a firewall). WS-RM offers greater reliability when using HTTP transport, but has a performance impact.

WS-RM is applicable only to HTTP transport. If you configure WS-RM on a message flow that uses JMS transport, the WS-RM settings are not used when the flow is deployed.

Systems that implement WS-RM retransmit messages that have not been successfully delivered and acknowledged, and prevent duplicate messages from being delivered to the application destination. WS-RM is a web services protocol and can be used with WS-Security and WS-Addressing.



Reliable messaging takes place between two endpoints known as the *reliable messaging source* and the *reliable messaging destination*. Before the messages are sent, the reliable source and reliable destination perform a message exchange to establish a *Sequence*. A Sequence is identified by a unique identifier and comprises a sequence of messages which are numbered starting from one. Sending a group of messages in a Sequence ensures the reliability of all the messages in that Sequence.

The reliable messaging source sends each message one or more times to the reliable messaging destination. The destination sends back acknowledgment for each message it receives to show that the message has been successfully received. If the reliable messaging source does not receive an acknowledgment that a message has been received by the destination, it sends the message again until an acknowledgment is received.

When all messages in a sequence have been successfully received by the destination and acknowledgment received by the source, the source sends a TerminateSequence message to instruct the destination that the message sequence is complete.

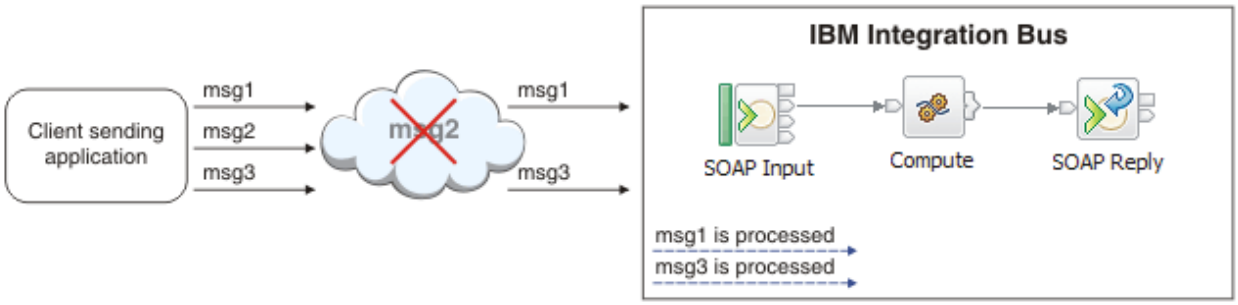
If the client is waiting for messages that have not been delivered, it can initiate a WS-MakeConnection request. WS-MakeConnection is a specification that describes how messages can be exchanged between a server and a client using a transport-specific back-channel. The client's MakeConnection request allows the server to respond with any queued messages that have not been received by the client.

IBM App Connect Enterprise does not support using HTTP compression or SSL with WS-RM.

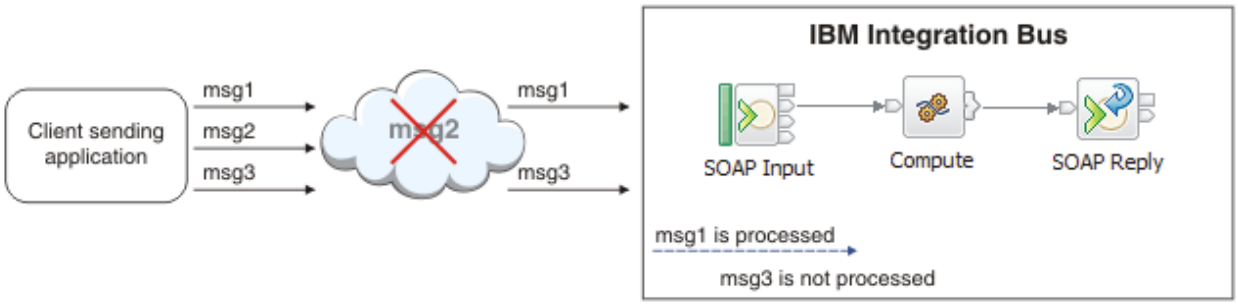
Message order

WS-RM specifies a number of delivery assurances to be supported by the provider. The InOrder assurance asserts that messages are delivered to the application destination by the RM Destination in the order in which they were sent, according to their sequence numbers. For example, if the RM Destination receives messages in the order m1, m3, m2, it first delivers m1, then retains m3 until it receives m2, and then delivers m2 and m3 to the application destination. Messages are not persisted in the event of an integration node restart.

The following diagram shows an example scenario where InOrder is not used, and messages need not be delivered in the order in which they were sent. The second message in the sequence is lost, but the third message is processed even though the second message has not been delivered.



In the following scenario the option is selected, and messages must be delivered in the order in which they are sent. The second message in the sequence is lost, and so the third message is not processed until msg2 can be delivered.



The InOrder assurance affects only inbound message processing, for example, when the policy set that asks for InOrder delivery is associated with a SOAPInput node or an inbound message flow. When InOrder delivery is enabled for inbound SOAP message processing, the messages are processed by the flow according to their message number as assigned by the RM Source.

The InOrder assurance does not apply to outbound message processing. If InOrder delivery is selected for outbound SOAP message processing, for example, with a SOAPRequest node, messages are not necessarily delivered to the application destination in the same order in which they are sent from the message flow.

Message flows for web services

Message flows that need to work with web services can use either the SOAP domain or one of the XML domains.

The following topics describe both types of flow.

- “SOAP domain message flows” on page 820
- “XML domain message flows” on page 826

The following topics describe web services tasks.

- “WS-Addressing overview” on page 830
- “WSDL overview” on page 837
- “Configuring message flows to process timeouts” on page 800
- “SOAP MTOM and the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes” on page 849

SOAP domain message flows

SOAP domain message flows typically use SOAP nodes, WSDL, and a common logical tree format that is independent of the exact format of the web service message.

The following nodes are provided for use in the SOAP domain:

- node
- node

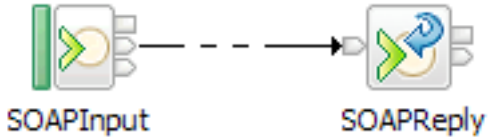
- node
- node
- node

The following nodes can also be used to simplify SOAP payload processing in a message flow; these nodes are not specific to the SOAP domain.

- node
- node

The SOAP nodes are used together in the following basic patterns:

- As a web service provider, for example:



- As a web service consumer, for example:

Or:

- As a web service facade, for example, by combining the provider and consumer scenarios:

You can use the SOAPExtract node in conjunction with these patterns to extract the SOAP payload. If you are working with the HTTP nodes, you can use the SOAPEnvelope node to rebuild a SOAP envelope.

The main SOAP domain nodes are typically configured by WSDL, and in this mode, a prerequisite for a SOAP domain message flow is a deployable WSDL. You can either import a WSDL file into an application or library, or generate a WSDL file from an existing message set. For more information about importing a WSDL file, see “Importing from WSDL” on page 2305. For more information about generating WSDL from an existing message set, see Message Sets: WSDL generation.

Deployable WSDL can then be used to configure SOAP nodes. You can do this by dragging the WSDL resource onto the node or by selecting the required WSDL resource from the node properties.

The WSDL is deployed with your completed message flow, enabling exceptions to be raised if a web service message does not correspond to the specified WSDL description.

You can change the operation mode of the SOAP domain nodes so that they act in gateway mode. In gateway mode, a WSDL is not required to configure the nodes because they handle generic request/

response and one-way SOAP messages that are not tied to a specific WSDL. For more details, see “Gateway operation mode for SOAP nodes” on page 827.

A client can send an HTTP GET request to the endpoint exposed by a SOAPInput node, suffixed with a query string `?wsdl`, and receive a response with the WSDL definition used to configure the flow; see “Message flow configuration with a WSDL file” on page 839.

The SOAP domain uses a common logical tree format that is independent of the exact format of the web service message. For details of the SOAP tree format, see “SOAP tree overview” on page 811. Useful WSDL information is included in the logical tree under `SOAP.Context`.

Example usage of WS-Addressing
Set up a sample message flow by using WS-Addressing, and test the flow.

About this task

Complete the steps in the following set of topics.

1. Build the main message flow that includes SOAP nodes, a Filter node, and a Mapping node by following the instructions in “Building the main message flow” on page 822.
2. Build a logging message flow, so that you can send a reply to an address different from the originating client, by following the instructions in “Building the logger message flow” on page 824.
3. Deploy the message flows by following the instructions in “Deploying the message flows” on page 825.
4. Test the message flows, by using a tool that uses HTTP. This task illustrates that the contents of the SOAP message determine where the replies are routed. Follow the instructions in “Testing the message flows” on page 825.

Building the main message flow
You can construct a sample main message flow to use with WS-Addressing.

About this task

These steps are the first in a set of instructions on setting up your system to use WS-Addressing with IBM App Connect Enterprise; they explain how to set up a message flow to use this feature. This topic describes how to construct a sample main message flow when using WS-Addressing.

Procedure

1. Switch to the Integration Development perspective.
2. Create message flow and message set projects using the **Start from WSDL and/or XSD files** wizard.
3. Select the **Web Services** folder on the message flow palette to display the contents, and drag a SOAPInput node onto the canvas.
4. Add a SOAPExtract node to the message flow to remove the SOAP envelope from the incoming message, followed by a SOAPReply node. Wire the Out terminal of the SOAPInput node to the In terminal of the SOAPExtract node, and wire the Out terminal of the SOAPExtract node to the In terminal of the SOAPReply node.
5. Select the WSDL file that you need under **Deployable WSDL** from the Active Working Set, and drag it onto the SOAPInput node.

   The SOAPInput node is configured with the WSDL.
6. Select the **Construction** folder on the message flow palette to display the contents.

7. Select a Trace node and move the mouse to the right of the SOAPExtract node.
   a) Click the left mouse button to add the node to the message flow.
      The name is selected automatically.
   b) Press **Enter** to accept the default name.
   c) Wire the submitPORequest terminal of the SOAPExtract node to the In terminal of the Trace node.
8. Select the Trace node to display the properties.
   a) Use the menu to set **Destination** to `File`
   b) Set the **File path** that you require.
   c) Enter the **Pattern** that you require.
9. Expand the **Routing** folder on the palette and select **Filter**.
10. Add the Filter node to the right of the Trace node.
    a) Type the name for the node that you require and press **Enter**.
    b) Wire the Out terminal of the Trace node to the In terminal of the Filter node.
11. Select the Filter node to display the properties.
    a) Enter the **Data source** name that you require.
    b) Change the name of **Filter expression** to the name that you selected for the Filter node.
    c) Clear the **Throw exception on database error** check box.
12. Double-click the Filter node to open the ESQL editor.
    Create or change the ESQL for the node; for more information, see "Creating ESQL for a node" on page 1618 and "Modifying ESQL for a node" on page 1620.
13. Expand the **Transformation** folder on the palette and select a Mapping node.
14. Add the Mapping node to the right of the Filter node.
    a) Type the name for the node that you require and press **Enter**.
    b) Wire the True terminal of the Filter node to the In terminal of the Mapping node.
    c) Wire the Out terminal of the Mapping node to the In terminal of the Reply node.
15. Select the Mapping node to display the properties, and change the name of the Mapping routine if required.
16. Double-click the Mapping node to open the mapping editor.
    a) Select **submitPORequest** as the map input.
    b) Select **SOAP_Domain_msg** as the map output.
    c) Click **OK**
    For more information, see Creating a message map from a Mapping node.
    The created map provides the default mapping for the Message Assembly Properties.
17. Expand the **SOAP_Domain_Msg** then **Body** in the output pane.
18. Right-click **Wildcard Message** in the output pane, then select **submitPORequest** on the input, and drag it to an element in the output below the body.
19. Set the transform type to **Submap** and then, in the list of properties, click **New..** to create the new submap.
20. In the New Map wizard for the submap, select **submitPORequest** as the input, and select **submitPOResponse** as the output.
21. Click **OK** to create the submap and enter the Graphical Data Mapping editor.
22. Expand the **submitPORequest** input tree and the **submitPOResponse** output tree and wire the required transforms from the input elements to the output.
23. Save the main map and the submap.

Building the logger message flow
This is the second of a set of instructions on setting up your system to use WS-Addressing with IBM App Connect Enterprise, and illustrates the use of a reply being sent to an address other than the originating client.

About this task

This topic describes the construction of a sample logger message flow when using WS-Addressing. This flow echoes back the input while creating a trace entry in a file to indicate that the flow has been invoked.

Procedure

1. Switch to the Integration Development perspective.
2. Select the message flow name that you used in "Building the main message flow" on page 822
3. Press the right mouse button and select **New->MessageFlow**.
   a) Enter the name that you require for this message flow; for example, `Logger`.
   b) Press **Finish** to create the flow.
4. Select the **HTTP** folder on the message flow palette to display the contents.
5. Select an **HTTPInput** node and move the mouse to the left side of the canvas.
   a) Click the left mouse button to add the node to the message flow and enter the name `Logger`.
   b) Press **Enter** to finish.
6. Select the **HTTPReply** node from the palette and move the mouse to the right of the HTTPInput node, leaving room for a node in between.
   a) Click the left mouse button to add the node to the message flow and enter the name that you require; for example, `Logger`.
   b) Press **Enter** to finish.
7. Select the **Construction** folder on the message flow palette to display the contents.
8. Select a **Trace** node and move the mouse to the right of the HTTPInput node.
   a) Click the left mouse button to add the node to the message flow and enter the name that you require; for example `Trace`.
   b) Press **Enter**.
   c) Wire the `out` terminal of the HTTPInput node to the `In` terminal of the Trace node.
   d) Wire the `out` terminal of the Trace node to the `In` terminal of the HTTPReply node.
9. Select the HTTPInput node to display the properties.
   In the **Basic** tab:
   a) Enter the **Data source** name that you require.
   b) Change the name of the input node `Logger` as the **Path suffix for URL**.
10. Select the **Input Message Parsing** tab and select `XMLNSC` as the **Message domain**.
11. Select the Trace node to display the properties.
    a) Set **Destination** to `File`
    b) Set the **File path** that you require.
    c) Enter the **Pattern** that you require.
12. Save the message flow.

Deploying the message flows
This is the third of a set of instructions on setting up your system to use WS-Addressing with IBM App Connect Enterprise, and illustrates the deployment of the message flows.

About this task

This topic describes the deployment of the message flows that you have already constructed.

Procedure

1. Select the **LocalProject** project under **BARs** in the navigator pane.
   a) Right-click in the Application Development view, then click **New** > **BAR file**.
   b) Set the location to `LocalProject`.
   c) In the Name field, enter the name that you selected for the main message flow described in "Building the main message flow" on page 822.
   d) Click **Finish** to open the BAR Editor.
2. In the BAR Editor, complete the following steps.
   a) Select **Message flows, libraries, and other message flow dependencies**, and select the main message flow and logger message flow, and the main message set.
   b) Click **Build and Save** and confirm that the build operation was successful.
3. To deploy the message flows to the default integration servers, right-click the BAR file that you have just built, then click **Deploy**.
4. Select the default integration server and click **Finish** to start the deployment.
   Ensure that you receive a successful response message, and press **OK** to dismiss the information dialog.
5. Use the **Deployment Log** to confirm that the deploy operation was successful:

Testing the message flows
This is the fourth of a set of instructions about how to set up your system to use WS-Addressing with IBM App Connect Enterprise; these instructions illustrate how to test the message flows.

About this task

This topic describes how to test the message flows that you have already constructed. In this scenario, you use a tool that uses HTTP protocol rather than IBM MQ protocol. You can use any tool that has the facilities that are described in the following procedure.

Procedure

1. Start the tool and select `http://localhost:nnnn/`, where *nnnn* is the address of the port that you are using.
2. Set the URL to the host, port, and selection for the deployed flow.
   The SOAPInput nodes listen on a different port from the HTTP nodes, and the listener is built into the integration server rather than using a different process.
3. The SOAPInput node expects a SOAPAction entry in the HTTP headers, therefore you must add one.
   a) Click **Add New Header**.
   b) Enter the **Value** part of the header.
      The value must match the `SOAPAction` attribute of the `SOAP:operation` element in your code.
   c) Select **New Header** in the **Name** pane.
   d) Change the name from **New Header** to `SOAPAction` and click **Enter**.
4. Select **Load File** and go to the directory that contains the XML file you want to use.

5. Select the file and click **Open**.

   Note the following conditions:

   - If the message does not include any WS-Addressing entries, the `ReplyTo` and `FaultTo` locations default to `anonymous`. This means that the results are returned on the original client connection.
   - If the message includes a WS-Addressing header (`ReplyTo`) with a value of `anonymous`, the reply is returned to the original client by using the original TCP/IP connection.
   - If the message includes a WS-Addressing header with a value of `FaultTo` explicitly included, the reply is returned to that address rather than the default of using the location that was specified in the `ReplyTo` header.

6. Click **Send** to test the flow.

   The result appears in the right pane.

XML domain message flows

If you are not using the SOAP domain, your message flow must take account of the bitstream format of the web service messages with which you are working. A different logical tree format is used by each domain.

If the messages are SOAP, you can use either the XMLNSC domain or the MRM XML domain. Both domains offer validation. The XMLNSC domain is more efficient, while the MRM XML domain can be useful if you have specific message transformation requirements (for example, if your message flow also uses binary data formats).

If the messages use MIME (for example, SOAP with Attachments or MTOM), you can use the MIME domain. In this case, your message flow typically needs to identify at least the MIME part that corresponds to the SOAP payload, then explicitly parse this part by using the XMLNSC or MRM domain.

In the SOAP domain, WSDL is used to configure your nodes automatically with the appropriate endpoint information. If you are not using the SOAP domain, select and configure the transport nodes manually. Typical WSDL bindings are:

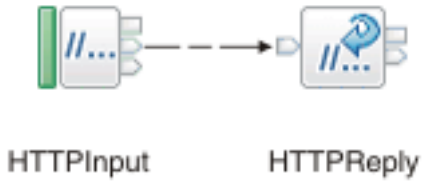
- SOAP/HTTP; in which case, implement a flow by using HTTP nodes. Use the HTTPInput and HTTPReply nodes if a flow implements a web service, or use the HTTPRequest node if a flow calls a web service.
- SOAP/JMS; where you implement a flow by using JMS or MQ nodes.

You can configure message flows that receive input messages from clients by using one transport, and interact with a web service or legacy application by using another.

You can propagate a message to more than one location. For example, the web service response to be returned to a client by an HTTPReply node might first be sent to an auditing application by using an MQOutput node, after making any required adjustments to the message headers.

Nodes are used together in the following basic patterns, by using HTTP nodes as example transports:

- As a web service provider, for example:



- As a web service consumer, for example:

- As a web service facade, for example:

If required, you can use the SOAPExtract and SOAPEnvelope nodes in conjunction with these patterns to respectively extract the SOAP payload and rebuild a SOAP Envelope.

To enable your message flow to validate messages, deploy an appropriate application, library, or message set with the flow. The application, library, or message set must contain a WSDL file. You can import a WSDL file into an application or library, or you can generate a WSDL from an existing message set. For details about importing existing WSDL, see "Importing from WSDL" on page 2305. For details about generating WSDL from an existing message set, see Message Sets: WSDL generation.

If you have generated a WSDL file from a message set, the generated message set contains message definitions for the relevant SOAP Envelope version and for the XML payload data defined by the WSDL. If you have imported a WSDL file into an application or library, message roots are created instead of message definitions. In the XMLNSC domain, messages can be validated against the message set, application, or library. In the MRM domain, messages can be validated against a message set only. For more details, see "Validating messages" on page 610.

Gateway operation mode for SOAP nodes
The SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes have two operation modes: WSDL mode, and Gateway mode.

If the node is configured to be in WSDL mode, which is the default, the web service operations performed by the node are specified by a WSDL that configures the node. In Gateway mode, SOAP nodes in a message flow handle generic SOAP request messages in the following scenarios.

Provider scenario
In a provider scenario, IBM App Connect Enterprise receives generic SOAP/HTTP or SOAP/JMS requests by using a SOAPInput node, and sends a reply to the originating client by using a SOAPReply node. A single SOAPInput node can receive any SOAP request message, and is not configured with a WSDL.

Consumer scenario
In a consumer scenario, IBM App Connect Enterprise can route a SOAP request to any external web service provider by using the SOAPRequest node or a pair of SOAPAsyncRequest and SOAPAsyncResponse nodes. Endpoint information can be specified in the local environment, which is used dynamically at run time to send the outbound request message.

Façade scenario
In a façade scenario, IBM App Connect Enterprise can receive SOAP requests from multiple different clients, and route them to any one of multiple back end web service providers, or existing systems. Endpoint information for the back-end web service provider can be specified dynamically at run time by setting values in the local environment.

The SOAP nodes act and are configured differently in the two different operation modes.

- If you configure a node in Gateway mode, some node properties are disabled because they are not applicable in Gateway mode.
  - The WSDL-specific properties are disabled when the node is switched to Gateway mode. The values are not cleared, and can be used if the node is switched back to WSDL mode.
  - The Response Message Parsing properties on the SOAPRequest node are disabled in Gateway mode.
  - Validation of the operation according to the WSDL does not take place because no WSDL exists against which to validate in Gateway mode. If a SOAPAsyncRequest and SOAPAsyncResponse

node pair is used in Gateway mode, the validation properties on the SOAPAsyncResponse node are disabled at run time, regardless of the validation settings in the BAR file.

  - No message set is associated with the node, and no schema is used. Therefore, no message validation takes place in Gateway mode. You can configure the flow downstream to enable validation.
  - Outbound MTOM is not supported in Gateway mode because validation cannot take place. However, you can enable validation, and therefore enable MTOM, by setting the Message Set in the local environment. For example:

    ```
    SET OutputRoot.Properties.MessageSet = 'myMessageSet';
    ```

    You might need to also enable validation on the Compute node or the SOAPReply node.
  - In the `User-defined SOAP headers` table on the SOAPInput node, all operations are set to `*`. The `WSDL-defined SOAP headers` table is cleared.

- Transport properties must be configured. The transport-specific properties are mandatory depending on the `Transport` property; for example, the JMS transport properties are mandatory if the selected transport is `JMS`.
- A SOAPInput node can only be configured to receive messages of a single specified transport, for example, HTTP. Use separate input nodes to send or receive messages with different transports.
- A SOAPRequest node can only be configured to send messages of a single specified transport, for example, JMS. However, you can change the transport for any message using the local environment.
- A SOAPAsyncRequest node can only be configured to send and receive messages over a single transport, for example, JMS, and this transport is always used for the paired SOAPAsyncResponse node to receive the response message. However, you can change the outbound request transport for any message by using the local environment. For example, if a SOAPAsyncRequest node is configured to use JMS transport, its paired SOAPAsyncResponse node always expects to receive responses over JMS, and this cannot be changed. At run time, the SOAPAsyncRequest node also then assumes JMS as the default transport. However, you can instruct it to instead send the request over HTTP by using the local environment. This request includes the WSA:ReplyTo JMS address for the SOAPAsyncResponse node.
- If a SOAPInput node receives a one-way SOAP request, the node attempts to detect that it is a one-way message. However, the node cannot detect all cases, and therefore it is sometimes necessary to instruct the SOAPReply node that it is a one-way message, by using the local environment. For example:

  ```
  SET OutputLocalEnvironment.Destination.SOAP.Reply.Gateway.OneWay = True;
  ```

  For more information, see “One-way messages in Gateway mode” on page 829.
- If you use the SOAPAsyncRequest node in Gateway mode, you must set the WS-Addressing `Action` property in the local environment in the message flow before the SOAPAsyncRequest node. Set this property by using `OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action`.
- If you use the SOAPRequest node in Gateway mode and the remote service provider expects a SOAPAction, set the SOAPAction in the flow. In Gateway mode the SOAPAction from a WSDL is not available to the node. For example, to set the SOAPAction using ESQL:

  ```
  SET OutputRoot.HTTPRequestHeader.SOAPAction = 'mySoapAction';
  ```

  By default the SOAPRequest node sends an empty SOAPAction of `""`.
- If you use the SOAPRequest node in Gateway mode and are using WS-Addressing, you must set the WS-Addressing `Action` property in the local environment in the message flow before the SOAPRequest node. Set this property by using `OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action`.
- In Gateway mode, you can add inbound "must understand" headers to the SOAPInput node or to the SOAPRequest and SOAPAsyncRequest nodes by specifying the details on the node property. However, if you will be adding services dynamically, where all the "must understand" headers cannot be known in advance, you can add these headers with a wildcard (`*`) for name, namespace, operation, or any combination of the three. This removes the need to redeploy your message flow when new services are

added. However, consider that if you add a wildcard for the name, namespace, and also operation, this means that all headers with a "must understand" flag are allowed into the flow.

- If you use the SOAPReply node as part of a façade message flow, with the SOAPInput node set to Gateway mode, and the SOAPRequest node acting in WSDL mode, disable validation on the SOAPReply node or add an explicit Message Set in the Properties folder as documented above. If you do not disable validation or reference a Message Set in the Properties folder, parsing errors occur when the message is serialized.
- In Gateway mode, the SOAP nodes send SOAP 1.1 messages by default, although they also accept inbound SOAP 1.2 messages. To send an outbound SOAP 1.2 message, set the `SOAP.Context` field to indicate that SOAP 1.2 is required. For example, to set this field using ESQL:

  ```
  SET OutputRoot.SOAP.Context.SOAP_Version = '1.2';
  ```

  The outbound message then uses a SOAP 1.2 SOAP Envelope. You can also set the namespace prefix used by the SOAP Envelope. For example, by using ESQL:

  ```
  DECLARE soapenc NAMESPACE 'http://www.w3.org/2003/05/soap-envelope';
  SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soap12 = soapenc;
  SET OutputRoot.SOAP.Context.SOAP_Version = '1.2';
  ```

  In this example, `soap12` is the prefix used in the outbound message.

One-way messages in Gateway mode
When you configure a SOAP node using WSDL, the WSDL specifies whether a given node operation is one-way or not. However, configuring a node in Gateway mode without WSDL means that this WSDL information is not available. Therefore, SOAP nodes configured as Gateways automatically attempt to detect one-way Operations based on the message content.

SOAPInput node one-way Operation detection

The SOAPInput node detects one-way messages in different ways depending on the transport used, and whether or not WS-Addressing is configured on the node.

- If the node uses HTTP transport and has WS-Addressing configured, the operation is determined to be one-way if the inbound message uses the special `WSA:None` address for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.
- If the node uses HTTP transport and does not have WS-Addressing configured, no auto detection of one-way messages takes place.
- If the node uses JMS transport and has WS-Addressing configured, the operation is determined to be one-way if either of the following conditions are true:
  - The inbound message uses the special `WSA:None` address (http://www.w3.org/2005/08/addressing/none) for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.
  - There is no JMS ReplyTo destination specified in the inbound message, and the inbound message uses the special `WSA:Anonymous` address (http://www.w3.org/2005/08/addressing/anonymous or http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous) for the `WSA:ReplyTo` and `WSA:FaultTo` addresses.
- If the node uses JMS transport and does not have WS-Addressing configured, the operation is determined to be one-way if there is no JMS ReplyTo destination specified in the inbound message.

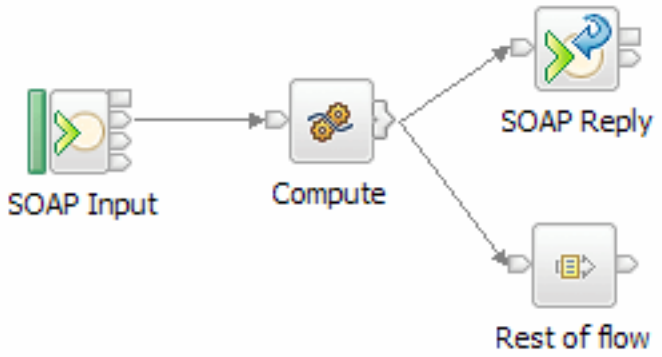
In Gateway mode, the `SOAP.Context.operationType` field is set to `GATEWAY` if the Operation is determined to be request-response, or `GATEWAY_ONE_WAY` if the Operation is determined to be one-way. If the operation is determined to be one-way, no reply is necessary or allowed. However, if the Operation is not determined to be one-way, the flow is configured with the assumption that the flow sends a reply. Therefore, if the Operation is one-way, you must specify that the Operation is one-way in order to allow the flow to free up resources, and, if HTTP is being used, to send an HTTP 202 acknowledgment back

to the originating client. Do this by setting the following field in the local environment before wiring the message to a SOAPReply node:

```
SET OutputLocalEnvironment.Destination.SOAP.Reply.Gateway.OneWay = 'true';
```

This setting instructs the SOAPReply node to complete the Message Exchange Pattern before sending an HTTP 202 acknowledgment, if required, and freeing up its resources.

One approach for using this setting would be in a Gateway flow like this:



In this flow, the Compute node determines if the message is one-way. If the message is one-way, the Compute node sets the local environment one-way setting, and sends a message to the SOAPReply node to complete the Message Exchange Pattern. If the flow is a Gateway flow and the one-way local environment option is set, any message received by the SOAPReply node causes it to ignore the message content and complete the Message Exchange Pattern. The flow can then continue through the other terminal of the Compute node.

It is not an error to send a message to the SOAPReply node with the one-way local environment option set if the message has been automatically determined to be a one-way message.

SOAPRequest node one-way Operation detection

In Gateway mode, the SOAPRequest node automatically detects whether a message is one-way only if WSA is used and the `WSA:ReplyTo` and `WSA:FaultTo` addresses are set to the special `WSA:None` address. To manually instruct the node that the message is one-way, set the following option in the local environment:

```
SET OutputLocalEnvironment.Destination.SOAP.Request.Gateway.OneWay = 'true';
```

For the SOAPRequest node, specifying that a message is one-way indicates to the node that a response is not expected, except for an HTTP 202 acknowledgment if HTTP transport is used. If JMS transport is used, it also allows the message to be sent under the control of any existing transaction, if the `Transaction mode` is to `Yes` or `Automatic`.

WS-Addressing overview
Use the following topics to learn more about WS-Addressing.

- "WS-Addressing" on page 831
- "WS-Addressing with the SOAPInput node" on page 831
- "WS-Addressing with the SOAPReply node" on page 832
- "WS-Addressing with the SOAPRequest node" on page 832
- "WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes" on page 833
- "WS-Addressing information in the local environment" on page 834

WS-Addressing
An overview of how you use WS-Addressing with IBM App Connect Enterprise.

Sending a message to an endpoint reference (EPR)

When sending a message to an endpoint reference, the following processes take place:

- The [destination] Message Addressing Property (MAP) is completed from the [address] property in the EPR.
- [reference parameters] are copied to top level SOAP headers from the [reference parameters] property of the EPR.
- The [action] property is required, but is not populated from the EPR.

  In this context, action is an absolute Internationalized Resource Identifier (IRI) that uniquely identifies the semantics implied by this message, and it must be the same as the HTTP SOAPAction if a non-empty SOAPAction is specified.
- The [message id] property must be specified if this message is part of a request-response Message Exchange Pattern (MEP); the message id is generated by default.

When replying with a non-fault message, the following processes take place:

- The EPR to reply to is selected from the [reply endpoint] MAP.
  - If this property contains the special address none, no reply is sent.
  - If this property contains the special address anonymous, the reply is sent on the return channel of the transport on which the request was received. This is the default value in the absence of any other supplied EPR.
  - Otherwise, the reply is sent to the [address] property in the Reply EPR,
- The [message id] property of the inbound message request is placed into the [relationship] property of the response, along with the predefined relationship of the reply part of the Universal Resource Identifier (URI) - which indicates that this message is a reply.

  For further information on the URI see Web Services Addressing URI Specification.
- A new [message id] property is specified for the reply, and this is generated by default.

WS-Addressing with the SOAPInput node
Various options are available when you use WS-Addressing with the SOAPInput node.

The SOAPInput node has a property for processing WS-Addressing information present in the incoming message called Use WS-Addressing.

If you select this property, the WS-Addressing information is processed and the process itself is called engaging WS-Addressing. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL, and this property is configurable from the WSDL, automatically by the IBM App Connect Enterprise Toolkit, when the WSDL is dropped onto the node. The behavior of the node when WS-Addressing is engaged or not engaged is as follows:

Addressing not engaged
No WS-Addressing processing is performed. If a message is received that contains any WS-Addressing headers they are ignored, and no WS-Addressing processing of any kind is performed, unless they are marked as MustUnderstand.

The inbound WS-Addressing headers in this case are visible in the message when it leaves the SOAPInput node under the Header folder of the SOAP parser in the message tree.

A fault is returned to the client if WS-Addressing headers exist in the incoming message, and they meet both of the following criteria:

- Marked as MustUnderstand
- Targeted at the role the SOAPInput node is operating in

Engaging WS-Addressing is how you instruct the node to 'understand' the WS-Addressing headers. In this case the WS-Addressing headers remain in the SOAP Header section of the SOAP parser, and no other SOAP node acts upon them. In all cases, they are treated as a SOAP header with no special meaning assigned to the WS-Addressing headers.

Addressing engaged:
WS-Addressing processing is performed as stated in the WS-Addressing specification. This processing means that messages that contain either submission addressing headers or final addressing headers are accepted.

A fault is returned if both submission addressing headers and final headers are present, and either of the following conditions is met:

- Neither is marked with a role.
- They are both marked with same role and the SOAPInput node is acting in that role.

Assuming that the WS-Addressing headers are valid and the `Place WS-Addressing Headers into LocalEnvironment` check box is selected on the SOAPInput node, all headers (including detectable inbound reference parameters) are removed from the inbound message tree and are placed into the local environment tree under the `SOAP.Input.WSA` folder. Moving the WS-Addressing headers to the local environment indicates that they have been processed by the integration node. The headers are removed from the message tree because they have been processed on input; otherwise they would not be valid if the message tree was sent out without further changes. They are stored in the local environment to allow you to inspect them.

Only reference parameters from the final specification are detectable because they have an attribute called `IsReferenceParameter` that allows them to be detected. Submission reference parameter headers do not have this attribute, therefore they are not detectable, and they are not moved into the local environment tree from the message tree.

You can change WS-Addressing reply headers before the SOAPReply node is reached. For more information about changing WS-Addressing information in the local environment, see "WS-Addressing information in the local environment" on page 834.

WS-Addressing with the SOAPReply node
Various options are available when you use WS-Addressing with the SOAPReply node.

The SOAPReply node uses WS-Addressing if WS-Addressing is engaged on the SOAPInput node that is referenced by the reply identifier of the message entering the reply node.

The SOAPReply node uses addressing information in the `Destination.SOAP.Reply.WSA` folder of the local environment to determine where to send the reply and with what Message Addressing Properties (MAPs).

If the `Destination.SOAP.Reply.WSA` does not exist, or is completely empty when inspected by the SOAPReply node, the node uses the default addressing headers that were part of the incoming message. Therefore, you do not have to propagate the local environment in the default case, and addressing still works as expected.

In the case where folders exist beneath the `Reply.WSA` folder, these folders are used to update the output message. Therefore, you can change, add, or remove parts of the default reply information generated by the input node, because any changes that you made to the tree are reflected in the outgoing message by the SOAPReply node. For details about WS-Addressing information in the local environment, see "WS-Addressing information in the local environment" on page 834.

If WS-Addressing is not engaged, this node does not perform any WS-Addressing processing.

WS-Addressing with the SOAPRequest node
Various options are available when you use WS-Addressing with the SOAPRequest node.

The SOAPRequest node has a property called `Use WS-Addressing`, for processing WS-Addressing information that is present in the incoming message.

If you select this property, the WS-Addressing information is processed and the process itself is called `engaging WS-Addressing`. The default is that WS-Addressing is not engaged.

You can also specify this property in the WSDL and this is configurable from the WSDL, automatically by the IBM App Connect Enterprise Toolkit, when the WSDL is dragged onto the node. The behavior of the node when WS-Addressing is engaged or not is as follows:

Addressing not engaged
The node does not add any WS-Addressing headers to the outgoing message, and does not process any WS-Addressing headers that might be present in the response message that is received by the node.

Addressing engaged:
The node first looks at the `Destination.SOAP.Request.WSA` folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing Message Addressing Properties (MAPs) in the outgoing message, by using the following default values:

- `Action`, from the WSDL configuration file. If this is not explicitly specified, this defaults to the value that is defined in the WSDL Binding specification.
- `To`, from the `Web Service URL` node property.
- `ReplyTo`, by using the special `Anonymous` address (assuming that the Operation being used is not a one-way message exchange program, in which case a `ReplyTo` by using the special `None` address is specified).
- `MessageID`, a unique UUID is used.

If the `Destination.SOAP.Request.WSA` folder in the LocalEnvironment is not empty, any user supplied MAPs override the default ones that were listed previously, on a property by property basis.

After the response to the request is received and if the `Place WS-Addressing Headers into LocalEnvironment` check box is selected on the SOAPRequest node, the SOAPRequest node removes all WS-Addressing headers from the response message and places them in the `SOAP.Response.WSA` folder. This folder allows you to query the headers in a similar manner to the way the SOAPInput node deals with the Input WS-Addressing headers.

WS-Addressing with the SOAPAsyncRequest and SOAPAsyncResponse nodes
The remote web service must understand WS-Addressing to be able to work with SOAPAsyncRequest and SOAPAsyncResponse nodes.

The SOAPAsyncRequest and SOAPAsyncResponse nodes require WS-Addressing; therefore, the remote web service must understand WS-Addressing to process the WS-Addressing headers that are sent from the SOAPAsyncRequest node, and to allow the response to be sent back to the corresponding SOAPAsyncResponse node, which is specified in the `address` property of the `ReplyTo` Message Addressing Property (MAP).

SOAPAsyncRequest node

The SOAPAsyncRequest node has a property called `Use WS-Addressing` that is read-only and has a default value of `true`, indicating that WS-Addressing is mandatory for this node. This property has the effect of permanently engaging WS-Addressing for this node and cannot be changed by the node, or by the WSDL that is used to configure this node.

The node first looks at the `Destination.SOAP.Request.WSA` folder in the local environment. If this folder is empty, the node automatically generates all required WS-Addressing MAPs in the outgoing message, using the following default values:

- `Action`, from the WSDL configuration file. If this value is not specified explicitly, the default value is defined by the WSDL Binding specification.
- `To`, from the `Web Service URL` node property.
- `ReplyTo`, the address of the corresponding SOAPAsyncResponse node.
- `MessageID`, a unique UUID is used.

If the Destination.SOAP.Request.WSA folder in the local environment is not empty, any user-supplied MAPs override the default ones listed previously on a property by property basis.

However, because of the nature of the SOAP asynchronous node pair, you cannot specify the address property of the ReplyTo Message Exchange Program (MEP), and this property is ignored if specified.

When the main MAPs are generated, the node looks in several places to obtain various pieces of context information to send in a <wmb:context> element under the ReferenceParameters section of the ReplyTo endpoint reference. If these locations exist and are not empty, the following additional information is added to the <wmb:context>:

- Destination.SOAP.Request.UserContext

  This information is added under a subfolder called UserContext.
- Destination.SOAP.Reply.ReplyIdentifier

  This information is added under a subfolder called ReplyID.

Use the user context to specify an arbitrary amount of data that will be sent with the message from the SOAPAsyncRequest node to the SOAPAsyncResponse node. By using the user context, you can pass state from one node to the other. Ensure that the amount of data that you send is small because this data is placed in the message.

Use the reply identifier to automatically correlate a SOAPInput node in the flow that contains the SOAPAsyncRequest node, with a SOAPReply node in the flow that contains the SOAPAsyncResponse node.

SOAPAsyncResponse node

After the response to the request is received, the SOAPAsyncResponse node can remove all WS-Addressing headers from the response message and places them in the SOAP.Response.WSA folder so that you can query the headers, if you select the node property Place WS-Addressing headers in local environment.

If the response message contains a user context that was specified by the SOAPAsyncRequest node, the user context is placed in the SOAP.Response.UserContext folder in the local environment.

If the response message contains a reply identifier that was specified by the SOAPAsyncRequest node, the reply identifier is placed in the Destination.SOAP.Reply.ReplyIdentifier folder in the local environment.

WS-Addressing information in the local environment
WS-Addressing header information can be placed in the local environment tree where it is visible to a message flow. WS-Addressing header information is only processed by the SOAP nodes.

Inbound messages

Inbound information is placed in the local environment by the SOAP node only if addressing is engaged on the node and you select the Place WS-Addressing Headers into LocalEnvironment property on the SOAPInput, SOAPAsyncResponse, or SOAPRequest nodes.

The following table describes the node specific WS-Addressing information in the local environment tree.

Node	Populates local environment property
SOAPInput	LocalEnvironment.SOAP.Input.WSA.*type*
SOAPAsyncResponse	LocalEnvironment.SOAP.Response.WSA.*type*
SOAPRequest	LocalEnvironment.SOAP.Request.WSA.*type*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" on page 835 section of this topic.

The local environment information for inbound messages is for your information only. If you engage addressing on the node, and select the `Place WS-Addressing Headers into LocalEnvironment` property on the node, WS-Addressing information is available for you to look at and use in your flow. The WS-Addressing properties are placed in the local environment after processing by the node. Note that the WS-Addressing folder and all its children are owned by an XMLNSC parser, therefore you can copy elements directly into any other tree that is owned by an XMLNSC parser. However, be aware that if you copy this folder (or any of its children) to a tree that is not owned by an XMLNSC parser, information in the tree is discarded unless you create an XMLNSC parser in the target tree first. This behavior can occur if you, for example, copy from the `InputLocalEnvironment` tree to the `OutputLocalEnvironment` tree.

Outbound messages

You can place outbound WS-Addressing header information in the local environment; however, this practice is necessary only to override the defaults that are generated by the node automatically . Outbound addressing headers are created only if WS-Addressing is enabled on the node.

The following table describes the node specific WS-Addressing information in the local environment tree that can be used to override the defaults for outbound messages.

Node	Populates local environment property
SOAPReply	`LocalEnvironment.Destination.SOAP.Reply.WSA.`*`type`*
SOAPRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*`type`*
SOAPAsyncRequest	`LocalEnvironment.Destination.SOAP.Request.WSA.`*`type`*

Where *type* is the structure of the subsection of the local environment WS-Addressing XML schema. For details about how the *type* maps to the WS-Addressing properties defined by the WS-Addressing specification, see the "Local environment property type" on page 835 section of this topic.

You can modify local environment information for outbound messages. The SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes generate default local environment settings that you can override. One exception to this table is that any attempt to override the WS-Addressing `ReplyTo` address on the SOAPAsyncRequest node is ignored.

For example, the following code shows how to set WS-Addressing information in the local environment for the SOAPRequest node. The WS-Addressing ReplyTo.Address and FaultTo.Address values should be entered as a single string, without line breaks.

```
 SET OutputRoot = InputRoot;
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.To.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.Address = 'jms:jndi:RESPONSEQ?
jndiConnectionFactoryName=QCF&
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.From.Address = 'jms:jndi:INPUTQ';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.Address = 'jms:jndi:RESPONSEQ?
jndiConnectionFactoryName=QCF&
jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&
jndiURL=file://C:/SOAPJNDIBindings';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.Action = 'http://WMB_BankImport/NewOperation';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.MessageID = 'test:my:msg:ID:1234578';
 SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.Version = 'Submission-2004/08';
```

Local environment property *type*

The local environment property *type* in the preceding tables corresponds to the WS-Addressing part of the local environment XML schema. The following table shows the corresponding message addressing properties (MAPs) of the WS-Addressing local environment schema for all nodes.

Element	Corresponds to abstract WS-Addressing MAP name
To	[destination endpoint]

Element	Corresponds to abstract WS-Addressing MAP name
From	[source endpoint]
ReplyTo	[reply endpoint]
FaultTo	[fault endpoint]
Action	[action]
MessageId	[message id]
RelatesTo	[relationship]
ReferenceParameters	[reference parameters]
Version	This element does not correspond to a MAP, but it is used to identify the version of WS-Addressing. The two main versions of WS-Addressing are Submission and Final. The default version that is used by all nodes is Final. Therefore, for outbound messages, set this element only if you want the version to be Submission. In the code example, the version is set to Submission by the inclusion of `'Submission-2004/08'` in the element. For incoming messages, this element is populated automatically with the version of the WS-Addressing headers that the inbound message used.

For more details about the message addressing properties defined by the WS-Addressing specification, see “What is WS-Addressing?” on page 816.

For outbound WS-Addressing, you can set an additional local environment property.

Element	Description
AddMustUnderstandAttribute	This element places the SOAP `mustUnderstand` attribute on each WS-Addressing header before the message is sent.

Example use of WS-Addressing information in the local environment
This example shows the setting of reference parameters in the local environment along with the corresponding messages as they appear on the wire.

In this example the web service exposes a simple ping operation.

ESQL to add reference parameters

The following ESQL example shows how to specify addressing headers in the local environment.

```
DECLARE Example_ns NAMESPACE 'http://ibm.namespace';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Parameter1 =
 'Integration node';
SET
 OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.ReplyTo.ReferenceParameters.Example_ns:Parameter2
 = 'Ping';

SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Parameter1 = 'Ping';
SET
 OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.Example_ns:Parameter2.
 (SOAP.NamespaceDecl)xmlns:Example_ns = 'http://ibm.namespace';
SET OutputLocalEnvironment.Destination.SOAP.Request.WSA.FaultTo.ReferenceParameters.gns:Parameter2 =
 'FAULT';
```

Request message

The following example is of an outgoing SOAP envelope in a message from a SOAPRequest node with `ReplyTo` and `FaultTo` reference parameters generated after using the above ESQL. It also shows the

other message addressing properties (MAPs) that are not set in the local environment, but are generated automatically by the node as a result of engaging WS-Addressing.

```
<NS1:Envelope xmlns:NS1="http://www.w3.org/2003/05/soap-envelope" xmlns:wsa="http://www.w3.org/2005/08/
addressing">
  <NS1:Header>
    <wsa:To>http://localhost:7801/Service</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">Ping</Example_ns:Parameter2>
        <Parameter1>Integration node</Parameter1>
      </wsa:ReferenceParameters>
    </wsa:ReplyTo>
    <wsa:FaultTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
      <wsa:ReferenceParameters>
        <Example_ns:Parameter2 xmlns:Example_ns="http://ibm.namespace">FAULT</Example_ns:Parameter2>
        <Parameter1>Ping</Parameter1>
      </wsa:ReferenceParameters>
    </wsa:FaultTo>
    <wsa:MessageID>urn:uuid:020C911C16EB130A8F1204119836321</wsa:MessageID>
    <wsa:Action>http://ibm.com/Service/Ping</wsa:Action>
  </NS1:Header>
  <NS1:Body>
    <NS2:Ping xmlns:NS2="http://ibm.com"></NS2:Ping>
  </NS1:Body>
</NS1:Envelope>
```

In the above example, reference parameters are set for the `ReplyTo` and `FaultTo` endpoint references (EPRs). If this message is sent to a SOAPInput node with WS-Addressing engaged, these `ReferenceParameters` are placed in the local environment of the flow that contains the SOAPInput node for use by the flow if the `Place WS-Addressing Headers into LocalEnvironment` property is selected. This option changes only what is placed in the local environment; it does not change the contents of the response message.

Response message

The following SOAP envelope is a response to the outgoing preceding message, as sent by a SOAPReply node. This example shows the MAP processing that happens automatically by the SOAPReply node. In this example, the `FaultTo` reference parameters are not present because the reply is not a SOAP fault. This response also shows where the reference parameters that belonged to the `ReplyTo` EPR appear in the response message.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://
www.w3.org/2005/08/
 addressing">
  <soapenv:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <Example_ns:Parameter2 wsa:IsReferenceParameter="true" xmlns:Example_ns="http://ibm.namespace">Ping</
Example_ns:
 Parameter2>
    <Parameter1 wsa:IsReferenceParameter="true">Integration node</Parameter1>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:Action>http://ibm.com/Service/PingResponse</wsa:Action>
    <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing/
reply">urn:uuid:020C911C16EB130A8F1204119836321
 </wsa:RelatesTo>
  </soapenv:Header>
  <soapenv:Body>
    <NS1:PingResponse xmlns:NS1="http://ibm.com">
      <NS1:PingResult>Ping</NS1:PingResult>
    </NS1:PingResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

WSDL overview

Use the following topics to learn more about WSDL files.

- "WSDL validation" on page 838

- “Message flow configuration with a WSDL file” on page 839
- “WSDL URI formats for JMS” on page 842
- “WSDL styles” on page 844
- “rpc-encoded SOAP messages” on page 845

WSDL validation
The WS-I Validator can be used to check your WSDL definitions against the Basic Profile.

For more information about the WS-I Basic Profile refer to the WS-I, and in particular the WS-I Basic Profile document:

- Web Services Interoperability Organization (WS-I)
- WS-I deliverables index

You can use the WS-I Validator to check your WSDL definitions against the Basic Profile; see WS-I Basic Profile Version 1.1.

You can run the validator in either of the following ways:

- Manually against a specific `.wsdl` resource in the workbench.

  This option enables you to investigate and fix WS-I compliance problems; all validation issues are displayed as task list errors and warnings.
- Automatically, when WSDL is imported or generated.

  You can import WSDL definitions by using the Message Model wizard.

You can control the behavior of the validator by setting a compliance level for your profile.

1. Select **Preferences** > **web services**.
2. Click **Service Policies**, and expand **Profile Compliance**.
3. Select one of the following profiles:
   - WS-I AP compliance level (WS-I Attachments Profile 1.0)
   - WS-I SSBP compliance level (WS-I Simple SOAP Binding Profile 1.0)
4. Select a compliance level:
   - Select **Suggest compliance** to run the validator with errors treated as unrecoverable, but warnings only notified. This is the default setting.
   - Select **Require compliance** to run the validator with errors and warnings treated as unrecoverable.
   - Select **Ignore compliance** if you do not want to run the validator.

The AP selection applies automatically to the SSBP field, therefore Ignore is not explicitly selectable unless the AP selection is set to Ignore.

The following terms refer to the three broad categories of WSDL definition:

- `document-literal` means the combination `style="document"` and `use="literal"`
- `rpc-literal` means the combination `style="rpc"` and `use="literal"`
- `rpc-encoded` means the combination `style="rpc"` and `use="encoded"`

The following are typical validation problems using the preceding terminology:

Your WSDL is rpc-encoded
WSDL with `use="encoded"` is not WS-I compliant and can lead to operational problems because products of different vendors can make different assumptions about the expected SOAP payload.

WS-I: (BP2406) The use attribute of a `soapbind:body`, `soapbind:fault`, `soapbind:header`, and `soapbind:headerfault` does not have the value of "literal".

Your WSDL is document-literal, but one or more WSDL part definitions refer to XML Schema types.
In `document-literal` WSDL, the SOAP body payload is the XML Schema element that is referred to by the appropriate WSDL part.

If a type is specified instead of an element, the SOAP payload is potentially ambiguous (the payload name is not defined) and interoperability problems are likely.

WS-I: (BP2012) A `document-literal` binding contains `soapbind:body` elements that refer to message part elements that do not have the element attribute.

Your WSDL is rpc-literal or rpc-encoded, but one or more WSDL part definitions refer to XML Schema elements.
In rpc-style WSDL, the SOAP body payload is the WSDL operation name, and its children are the WSDL parts that are specified for that operation.

If an element is specified instead of a type, the SOAP message payload is potentially ambiguous (the payload name might be the WSDL part name or the XML Schema element name), and interoperability problems are likely.

WS-I: (BP2013) An rpc-literal binding contains `soapbind:body` elements that refer to message part elements that do not have the type attribute.

Your WSDL includes SOAP header, headerfault or fault definitions that refer to XML Schema types.
In rpc-style WSDL, the SOAP body is correctly defined through XML Schema types as described above.

SOAP headers and faults, however, do not correspond to an rpc function call in the same way as the body.

In particular, there is no concept of 'parameters' to a header or fault, and a header or fault must always be defined in terms of XML Schema elements to avoid potential ambiguity. Effectively, header and fault definitions in WSDL are always `document-literal`.

WS-I: (BP2113) The `soapbind:header`, `soapbind:headerfault`, or `soapbind:fault` elements refer to `wsd:part` elements that are not defined using only the "element" attribute.

Your WSDL is rpc-literal or rpc-encoded, but no namespace was specified for an operation.
In rpc-style WSDL, the SOAP message payload is the WSDL operation name, qualified by a namespace that is specified as part of the WSDL binding.

If no namespace is specified then the SOAP message payload is potentially ambiguous (the payload name might be in no namespace, or might default to use a different namespace, such as the target namespace of the WSDL definition) and interoperability problems are likely.

WS-I: (BP2020) An rpc-literal binding contains `soapbind:body` elements that either do not have a namespace attribute, or have a namespace attribute value that is not an absolute URI.

Your WSDL includes a SOAP/JMS binding.
If your WSDL uses a SOAP/JMS transport URI it is not WS-I compliant. An error is shown if strict WS-I validation is enabled. To disable strict WS-I validation, click **Window** > **Preferences** > **Integration Development** > **WSDL** > **Validation** and select the **WS-I BP 1.1: Allow SOAP/JMS as transport URI**. By default strict WS-I validation is disabled.

Web service interoperability is improved if you implement the following actions:

- Use document-style WSDL whenever possible.
- Use literal encoding, if rpc-style WSDL is necessary.
- Ensure that the WSDL operation definitions are qualified by a valid namespace attribute, if rpc-encoded WSDL must be used.

Message flow configuration with a WSDL file
You can use a WSDL file to configure message flows.

Message flows that work with web services typically use the SOAP nodes. For details about the SOAP nodes, see “IBM App Connect Enterprise and web services” on page 806. You can change the operation mode of the SOAP nodes so that they act in gateway mode. In gateway mode, a WSDL file is not required

to configure the nodes because they handle generic request and response and one-way SOAP messages that are not tied to a specific WSDL file.

You can create a web service in IBM App Connect Enterprise in the following ways:

- By importing a WSDL file.
- By dragging a WSDL file onto a SOAP node to configure that node.
- By specifying the WSDL file in the `WSDL file name` property of a SOAP node.
- By defining a new interface with the **New Integration Service** wizard.

When you drag a WSDL file onto a SOAP node, the node properties are configured from the properties in the WSDL `address` URI. The transport properties on the SOAP node are populated according to the first binding imported from the WSDL file. Therefore, if the first imported binding describes a JMS transport, the JMS Transport properties are populated; if the first imported binding describes an HTTP transport, the HTTP Transport properties are populated. If you select another imported binding, the transport properties are populated accordingly. The `portType` appears differently depending on the transport selected.

The WSDL `address` element URI can exist in two different formats, W3C format, or IBM (deprecated) format. The format of the WSDL URI affects the names of the WSDL properties that the parser looks for to populate the SOAP node properties. For example, the `JNDI context parameters` table is not populated when you import an IBM-style WSDL because it does not support these properties in the WSDL address URI. The table is populated only if JNDI context parameters are present in a W3C-style WSDL. For details, see "WSDL URI formats for JMS" on page 842.

If you supply a service definition, endpoint properties are set automatically, but you can also set or override these properties manually.

Optionally, WSDL definitions can be split into multiple files. The typical arrangement is that a top-level service definition file imports a binding file, the binding file imports an interface file, and this interface file imports or includes schema definition files.

A WSDL `portType` (the logical WSDL interface) is not sufficient on its own to configure a SOAP node; a specific binding is required so that the SOAP payload is well-defined at run time.

A binding defines a use, which can be `document` (the default) or `rpc`. If the use is `document`, the SOAP payload is described by an XML Schema element in the WSDL. If the use is `rpc`, the SOAP payload is the WSDL operation name in a specified namespace.

If you drag a WSDL file onto a SOAP node from a shared library, the `WSDL file name` and `Message model` properties on the node specify the name of the shared library.

Configuring the SOAP nodes

The following nodes are configured explicitly by WSDL files:

- node
- node
- node

The following nodes are configured implicitly by WSDL files, because they inherit the WSDL configuration of the node with which they are paired:

- node
- node

A SOAPReply node is always used with a SOAPInput node.

A SOAPAsyncResponse node is always used with a SOAPAsyncRequest node, associated by the `Unique Identifier` property.

Using WSDL with the SOAPAsyncRequest node
A SOAPAsyncRequest node must be associated with a WSDL file unless it is operating in gateway mode.

About this task

In WSDL mode, when you select a WSDL file for the `WSDL file name` field on the SOAPAsyncRequest node, the WSDL file is validated to ensure that it is WS-I compliant. If the WSDL file uses a SOAP/JMS transport URI, it is not WS-I compliant, but by default no error is shown. To enable strict WS-I validation and display a warning when a SOAP/JMS transport is used, click **Window** > **Preferences** > **Integration Development** > **WSDL** > **Validation** and clear the **WS-I BP 1.1: Allow SOAP/JMS as transport URI** check box.

After a valid WSDL file is selected, the project that contains the WSDL file is added as a referenced project to the corresponding application or library, if the reference does not exist. To use a WSDL file in a shared library, the container that contains your message flow must reference that shared library. If the WSDL file is not valid, or an incorrect file name is entered, an error message is displayed in the Properties view and all WSDL properties are blank.

If the SOAPAsyncRequest node is created by dragging a WSDL file onto the Message Flow editor, this property is preset to the name of the WSDL file. If the name of the WSDL file is not preset, you can set this property in one of the following ways.

Procedure

- If you have Deployable WSDL, you can select from the Deployable WSDL files by clicking **Browse**.
- If you have WSDL definitions, but no message set, then you can create a message set:
  a) Click Browse to open the WSDL Selection window.
  b) Click **Import/Create New** to open the **Import WSDL file** wizard.
  c) Enter the message set name and message set project name. Click Next.
  d) Select the relevant option:
     – If your WSDL file exists in your workspace, select **Use resources from the workspace**, and select the WSDL file.
     – If your WSDL file is in the file system, select **Use external resources**. Select the WSDL file. Click **Next**.
  e) Select the WSDL bindings to import. Any warnings or errors are displayed in the wizard banner.
  f) Click **Finish**. Result: Creates a new message set project and message set, with message definitions. The WSDL definitions are added to the Deployable WSDL folder.
  g) You can now select the WSDL file from the **WSDL Selection** window. Click **OK**.
- If you have a message set but no WSDL definition, you must generate a WSDL definition. See "Message Sets: Generating a WSDL definition from a message set" on page 2317.
- Drag a WSDL file from a message set onto the node.
- Type in a file name that is relative to the message set project in which the deployable WSDL file exists.

Results

When you save the flow file, it is validated that the `WSDL file name` exists in the message set. If it is does not, an error is generated, and you will not be able to add a flow that contains this SOAPAsyncRequest node to the BAR file.

Querying WSDL with ?wsdl
You can interrogate web services using ?wsdl.

A web service client can send an HTTP GET request with a `?wsdl` query string to an IBM App Connect Enterprise web service, and receive a representation of the WSDL that was used to configure the input node that provides the endpoint for the service. You can do this only for input nodes that use HTTP and

not JMS transport. The protocol of the HTTP GET request must match the protocol of the flow; therefore if the flow uses SSL, the HTTP GET request must begin `https://`.

The client starts by sending a simple `?wsdl` query and retrieves the complete WSDL definition by following a chain of referenced imports or includes. For example, if the web service endpoint is `http://localhost:7800/test1`, the initial client request is:

```
GET http://localhost:7800/test1?wsdl
```

This request returns the top-level WSDL service definition, which might include imports for further sections of the WSDL definition. For example, if the returned WSDL has a line:

```
<wsdl:import ... location="http://localhost:7800/test1?wsdl=wsdl0"/>
```

then the client sends a corresponding request to retrieve that section of the WSDL:

```
GET http://localhost:7800/test1?wsdl=wsdl0
```

One or more WSDL sections can also have imports for XML Schema data, for example:

```
<xsd:import ... schemaLocation="http://localhost:7800/test1?xsd=xsd0"/>
```

The client again sends a corresponding request to retrieve that data:

```
GET http://localhost:7800/test1?xsd=xsd0
```

Only semantically correct references can be followed:

- `<wsdl:import>` elements that are immediate children of `wsdl:definition` elements
- `<xsd:import>` and `<xsd:include>` elements that are immediate children of `xsd:schema` elements

where `wsdl` is a shorthand (namespace prefix) for `http://schemas.xmlsoap.org/wsdl/`, and `xsd` is a shorthand for `http://www.w3.org/2001/XMLSchema`. A request made to a URI from an element which superficially looks like an `<import>` or `<include>`, for example, an element in a comment, results in a SOAP Fault being returned. Only the simple `?wsdl` query string, and subsequent queries that exactly match those queries specified in semantically valid imports and includes, result in data being returned.

The WSDL definition returned is logically equivalent to the deployable WSDL in the IBM App Connect Enterprise Toolkit, with inline schemas externalized. It might not be physically identical to the original imported WSDL definition. Although a SOAPInput is configured with a specific WSDL binding, the WSDL returned also includes other bindings that are not used by the flow if these were part of the original WSDL definition that was imported.

WSDL URI formats for JMS

You must use WSDL to configure SOAP nodes. When using WSDL with a JMS transport, different URI formats can exist in the `address` element in the WSDL, which affect how properties are parsed and applied to the configured nodes.

Two different URI formats can exist in the WSDL `address` element. Several node properties are initially set from properties in the imported WSDL, which is parsed according to which type of URI is found in the WSDL element. The first type is the W3C SOAP JMS specification format. For example:

```
<soap:address location="jms:jndi:REPLYTOQ2?jndiConnectionFactoryName=QCF&amp;
 jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&amp;
 jndiURL=file:/C:/mqsi6/webservices/SOAP/JMS/JNDI&amp;
 targetService=SOAPJMSGenMessageSetSOAP_JMS_Service&amp;
 timeToLive=30000"
/>
```

The second URI format for the `address` element is a proprietary IBM format which is currently deprecated. For example:

```
<soap:address location="jms:/queue?destination=jms/RequestQ&amp;
 connectionFactory=jms/WMBQCF&amp;
```

```
  targetService=SOAPJMSGenMessageSetSOAP_JMS_Service&amp;
  initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory&amp;
  jndiProviderURL=file:/C:/mqsi6/webservices/SOAP/JMS/JNDI"
 />
```

There are several differences between these URI formats. IBM App Connect Enterprise accepts both URI formats. Different WSDL properties are used to set the SOAP node properties depending on which URI format is used in the WSDL address element.

The following table shows how WSDL properties are parsed into SOAPInput node properties. The columns headed "W3C names in URI" and "W3C allowed values" indicate the property names that the parser looks for when a W3C-style URI is found, and the allowed values for those properties. The columns headed "IBM names in URI" and "IBM allowed values" indicate the property names that the parser looks for when an IBM-style URI is found in the WSDL, and the allowed values for those properties. Where more than one property name is shown in a table cell, the node property is set to the value of the first of those property names found in the WSDL `address` element. Any properties found in the WSDL `address` element that are not parsed into node properties are discarded.

SOAPInput node property name	W3C SOAP/JMS specification names	W3C names in URI	W3C allowed values	IBM names in URI	IBM allowed values
Source	`soapjms:destinationName`	`jms-name` (in URI)	`<string>`	`destination`	`<string>`
Connection factory name	`soapjms:jndiConnectionFactoryName`	`jndiConnectionFactoryName`	`<string>`	`connectionFactory`	`<string>`
Initial context factory	`soapjms:jndiInitialContextFactory`	`jndiInitialContextFactory`	`<string>`	`initialContextFactory`	`<string>`
JNDI URL bindings location	`soapjms:jndiURL`	`jndiURL`	`<URL>`	`jndiProviderURL`	`<URL>`
JNDI parameters	`soapjms:jndiContextParameter`	`name=value`	`<string for name and value>`	N/A	`<string>`
Delivery mode	`soapjms:deliveryMode`	`deliveryMode`	`NON_PERSISTENT` `PERSISTENT`[1]	`deliveryMode` `persistence`	`<int 1 | 2>`
Message priority	`soapjms:priority`	`priority`	`<int 0-9>`	`priority` `Priority`	`<int 0-9>`
Target service	`soapjms:targetService`	`targetService`	`<string>`	`targetService`	`<string>`

Notes:

1. IBM App Connect Enterprise accepts the values 1 and 2 when parsing a W3C-style URI for compatibility reasons, but the W3C specification allows only the string values `NON_PERSISTENT` and `PERSISTENT` for this property.

The following table shows how WSDL properties are parsed into SOAPRequest andSOAPAsyncRequest node properties. The columns headed "W3C names in URI" and "W3C allowed values" indicate the property names that the parser looks for when a W3C-style URI is found, and the allowed values for those properties. The columns headed "IBM names in URI" and "IBM allowed values" indicate the property names that the parser looks for when an IBM-style URI is found in the WSDL, and the allowed values for those properties. Where more than one property name is shown in a table cell, the node property is set to the value of the first of those property names found in the WSDL address element. Any properties found in the WSDL `address` element that are not parsed into node properties are populated in the `User Parameters` table.

SOAPRequest or SOAPAsyncRequest node property name	W3C SOAP/JMS specification names	W3C names in URI	W3C allowed values	IBM names in URI	IBM allowed values
Destination	soapjms:destination	jms-dest (in URI)	<string>	destination	<string>
Connection factory name	soapjms:jndiConnectionFactoryName	jndi-ConnectionFactoryName	<string>	connectionFactory	<string>
Initial context factory	soapjms:jndiInitialContextFactory	jndi-InitialContextFactory	<string>	initialContextFactory	<string>
JNDI URL bindings location	soapjms:jndiURL	jndiURL	<URL>	jndiProviderURL	<URL>
JNDI parameters	soapjms:jndiContextParameter	jndi-Parameter name=value	<string for name and value>	N/A	<string>
Delivery mode	soapjms:deliveryMode	deliveryMode	NON_PERSISTENT PERSISTENT[1]	deliveryMode persistence	<int 1 | 2>
Message expiration	soapjms:timeToLive	timeToLive	<int>	timeToLive	<int>
Message priority	soapjms:priority	priority	<int 0-9>	priority Priority	<int 0-9>
Reply to destination	soapjms:replyToName	replyToName	<string>	replyToName replyTo replyToDestination replyDestination	<string>
Target service	soapjms:targetService	targetService	<string>	targetService	<string>
User parameters	UserProperties	<any other property name>	<string>	<any other property name>	<string>

Notes:

1. IBM App Connect Enterprise accepts the values 1 and 2 when parsing a W3C-style URI for compatibility reasons, but the W3C specification allows only the string values NON_PERSISTENT and PERSISTENT for this property.

WSDL styles

The SOAP nodes are configured by using a specific WSDL binding that has a style of either document (the default) or rpc. All operations that are defined in a specific WSDL binding are usually defined with the same use, which can be either literal (the default) or encoded.

The following terms are used to describe the three general types of WSDL bindings:

- document-literal (style="document", use="literal")
- rpc-literal (style="rpc", use="literal")
- rpc-encoded (style="rpc", use="encoded")

The shape of the runtime SOAP message that is defined by the WSDL file depends on the binding type, as shown in the following table:

WSDL binding type	Description
`document-literal`	The SOAP payload is described by XML schema. The `wrapped document-literal` convention constructs the XML schema so that the first child of the SOAP Body matches the operation name.
`rpc-literal`	The SOAP payload is described by the WSDL (operation and part name) and then by XML schema.
`rpc-encoded`	The SOAP payload has the same general shape as `rpc-literal`, but can carry SOAP encoding annotations designed to give the receiver additional information about the message that is sent. The annotations are slightly different between SOAP 1.1 and SOAP 1.2.

All three WSDL styles are supported by IBM App Connect Enterprise. The best style for a new service is `document-literal`, and it is also the least likely to cause interoperability problems. You can use the `wrapped document-literal` convention to explicitly associate the SOAP payload with the operation.

Both `document-literal` and `rpc-literal` are WS-I compliant. The `rpc-encoded` style is not WS-I compliant and can cause interoperability problems if the web service client and server use different technologies. Some common issues encountered with WSDL styles are described in "WSDL validation" on page 838.

It is simple to create and parse SOAP messages described by `document-literal` or `rpc-literal` WSDL, because the payload is standard XML described by the message set created from the WSDL.

For more information, see "rpc-encoded SOAP messages" on page 845.

rpc-encoded SOAP messages

The SOAP nodes are configured using a specific WSDL binding that has a style of either `document` (the default) or `rpc`. All the operations defined within a particular WSDL binding are usually defined with the same use, which can be either `literal` (the default) or `encoded`.

SOAP messages that are `rpc-encoded` can carry SOAP encoding annotations that are designed to give the receiver additional information about the message being sent. The following four types of annotations are common:

- xsi:type
- encodingStyle
- Arrays
- Multi-reference

Arrays and multi-reference elements can cause interoperability problems, and require specific intervention by the message flow developer, as follows:

- If the message flow builds outbound messages incorporating SOAP arrays, use ESQL to add any required attributes, as follows:
  - `arrayType` for SOAP 1.1
  - `arraySize` and `itemType` for SOAP 1.2
- If the message flow receives messages using SOAP arrays, disable validation.
- If the message flow receives messages using multi-reference encoding, disable validation and, if necessary, navigate the resulting logical tree using the `href` (or `ref`) and `id` attributes.

xsi:type

An `xsi:type` attribute can be added to an element to specify its type. For example:

```
<data xsi:type="xsd:string">text</data>
```

where the namespace prefix `xsi` is defined as follows:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

For a web service built from WSDL and XML schema, the type information is already available and the `xsi:type` is redundant.

Generally, when building an outbound message, you do not add `xsi:type` information. The following ESQL example shows how to add `xsi:type` information if it is required:

```
DECLARE xsd NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE xsi NAMESPACE 'http://www.w3.org/2001/XMLSchema-instance';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsi = xsi;

SET OutputRoot.SOAP.Body.rpc:op1.part1.data.(SOAP.Attribute)xsi:type = 'xsd:string';
```

When parsing an inbound message, any `xsi:type` information is added to the logical tree in the same way as other attributes, as follows:

```
(0x03000000:PCDataField):data = 'text' (CHARACTER)
(
  (0x03000100:Attribute)http://www.w3.org/2001/XMLSchema-instance:type =
 'xsd:string' (CHARACTER)
)
```

encodingStyle

An `encodingStyle` attribute can be added to an element to specify the SOAP encoding style used. For example:

```
<tns:op1 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

where the namespace prefix `soapenv` is defined as the namespace of your SOAP Envelope, as follows:

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" (SOAP 1.1)
xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"   (SOAP 1.2)
```

The value itself is a list of URIs, but the values in common use are as follows:

```
"http://schemas.xmlsoap.org/soap/encoding/" (SOAP 1.1)
"http://www.w3.org/2003/05/soap-encoding"   (SOAP 1.2)
```

In SOAP 1.1, the `encodingStyle` attribute can be added to any element. In SOAP 1.2 the `encodingStyle` attribute can be added only to children of Body, Header and Detail.

Generally, you build an outbound message, you do not add the `encodingStyle` attribute. The following ESQL example shows how to add `encodingStyle` information, if it is required:

```
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';

SET OutputRoot.SOAP.Body.tns:op1.(SOAP.Attribute)soapenv:encodingStyle = soapenc;
```

When parsing an inbound message, any `encodingStyle` attributes are added to the logical tree in the same way as other attributes, as follows:

```
(0x03000100:Attribute)http://schemas.xmlsoap.org/soap/envelope/:encodingStyle =
```

```
     'http://schemas.xmlsoap.org/soap/encoding/' (CHARACTER)
```

Arrays

A SOAP array is an element containing a sequence of child elements of the same type. In the following XML schema example, there is a type called *data* with two elements: a simple string and an array. In the schema, the field called *array* has an unspecified number of children of type string. The name of those child elements is not specified.

```
<xsd:complexType name="ArrayOfString">
  <xsd:complexContent mixed="false">
    <xsd:restriction base="soapenc:Array">
 <xsd:attribute wsdl:arrayType="xsd:string[]" ref="soapenc:arrayType"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="data">
  <xsd:sequence>
     <xsd:element name="simple" type="xsd:string"/>
     <xsd:element name="array"  type="tns:ArrayOfString"/>
  </xsd:sequence>
</xsd:complexType>
```

The namespaces used in the example are from WSDL 1.1 and SOAP 1.1, as follows:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

The following example is part of a valid instance document matching the schema:

```
<array soapenc:arrayType="xsd:string[]">
  <item>item1</item>
  <item>item2</item>
</array>
```

When you build an outbound message, you must add the appropriate attributes. For example, the following ESQL shows how to add the `arrayType` attribute:

```
DECLARE xsd     NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';

SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;

SET OutputRoot.SOAP.Body.rpc:op1.p1.array.(SOAP.Attribute)soapenc:arrayType = 'xsd:string[]';
```

The attributes used by SOAP 1.1 and SOAP 1.2 are different. SOAP 1.1 uses the `arrayType` attribute. The size of the array can be specified, but is not required. SOAP 1.2 uses two separate attributes. The equivalent SOAP 1.2 attributes for the previous example are `soapenc:itemType="xsd:string"` and `soapenc:arraySize="2"`, where the namespace prefixes are defined as follows:

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc=" http://www.w3.org/2003/05/soap-encoding"
```

You must disable validation if your message flow receives SOAP encoded messages containing SOAP arrays. The instance document cannot be validated against the schema when parsing an inbound message, because the name of the array items is not defined by the schema.

Multi-reference

The following example shows a SOAP 1.1 request message for a WSDL `rpc-encoded` operation called op1:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Body   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
                  xmlns:rpc="http://example/rpc">
    <rpc:op1>
       <p1>
         <simple>text</simple>
         <array soapenc:arrayType="xsd:string[]">
           <Item>item1</Item>
           <Item>item2</Item>
         </array>
       </p1>
    </rpc:op1>
  </soapenv:Body>
</soapenv:Envelope>
```

A SOAP implementation can reorganize this logical message to use multi-reference elements, as follows:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <soapenv:Body   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:rpc="http://example/rpc">
    <rpc:op1>
      <p1 href="#id1"/>
    </rpc:op1>
    <rpc:data id="id1">
      <simple>text</simple>
      <array href="#id2"/>
    </rpc:data>
    <soapenc:Array id="id2" soapenc:arrayType="xsd:string[]">
      <Item>Array Element 0</Item>
      <Item>Array Element 1</Item>
    </soapenc:Array>
  </soapenv:Body>
</soapenv:Envelope>
```

The message is logically equivalent to the first example, but the children of elements `p1` and `array` have been split out into separate sibling elements and then referenced using the `href` attribute.

The introduced elements, such as `<data>`:

- Can be referenced from more than one location, which allows the message to state that such elements are identical, as opposed to separate items which have the same value. The message size could be reduced if a shared element is large and referenced many times.
- Can directly or indirectly reference themselves, which allows the message to represent a graph that contains circular references.

Neither of these considerations applies to the previous example.

Generally, when you build an outbound message, you do not encode multi-reference elements unless the message represents a graph. Otherwise, the multi-reference encoding is optional. The following ESQL example shows how to encode multi-reference elements:

```
-- ESQL namespace prefixes
DECLARE soapenc NAMESPACE 'http://schemas.xmlsoap.org/soap/encoding/';
DECLARE xsd     NAMESPACE 'http://www.w3.org/2001/XMLSchema';
DECLARE rpc NAMESPACE 'http://example/rpc';

-- define XML namespace prefixes
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:soapenc = soapenc;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:xsd = xsd;
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:rpc = rpc;

-- build request message
SET OutputRoot.SOAP.Body.rpc:op1.p1.(SOAP.Attribute)href = '#id1';

SET OutputRoot.SOAP.Body.rpc:data.(SOAP.Attribute)id = 'id1';
SET OutputRoot.SOAP.Body.rpc:data.simple = 'text';
SET OutputRoot.SOAP.Body.rpc:data.array.(SOAP.Attribute)href = '#id2';

SET OutputRoot.SOAP.Body.soapenc:Array.(SOAP.Attribute)id = 'id2';
SET OutputRoot.SOAP.Body.soapenc:Array.(SOAP.Attribute)soapenc:arrayType = 'xsd:string[]';
SET OutputRoot.SOAP.Body.soapenc:Array.Item[1] = 'item1';
SET OutputRoot.SOAP.Body.soapenc:Array.Item[2] = 'item2';
```

When parsing an inbound message, multi-reference elements do not match the XML schema from the web service WSDL. If validation is enabled on a SOAP node, then an exception is thrown. If you use multi-reference encoding, then you must disable validation.

When you have a logical tree built from the message, you can propagate this tree to another SOAP node for output. For example, with a façade flow as shown in the following diagram, if the SOAPRequest node receives a response using multi-reference encoding, you can propagate the logical tree for the response to the SOAPReply node, and a SOAP message with multi-reference encoding is returned to the original client.

The original client must also understand the multi-reference encoding.

To manipulate the logical tree for a multi-reference encoded message, navigate the `href` and `id` attributes in ESQL. These attributes are slightly different in SOAP 1.1 and SOAP 1.2, as follows:

- In SOAP 1.1, the referencing element has an attribute `href="#id"` and the item referenced `id="id"`.
- In SOAP 1.2, the referencing element has an attribute `ref="id"` and the item referenced `id="id"`.

SOAP MTOM and the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes
The use of outbound MTOM messages can be configured on the SOAPReply, SOAPRequest, and SOAPAsyncRequest nodes.

The nodes have a property called `Allow MTOM`, which defines whether MTOM can be used.

An MTOM output message is written if all of the following criteria are met:

- The `Allow MTOM` property is selected on the **WS Extensions** tab.
- Validation is enabled. The `Validate` property on the SOAPRequest and SOAPAsyncRequest nodes controls validation of the anticipated response message and not validation of the outgoing request. MTOM output is therefore suppressed unless you set `Validate` to `Content and value` on a preceding input node or transformation node.
- No child elements exist below `SOAP.Attachment` in the logical tree. If child elements are present, SOAP with Attachments (SwA) is used.
- Elements exist in the output message that are identified as base64Binary in the associated XML Schema and whose length does not fall below a default threshold size of 1000 bytes.

You can use the local environment setting `MTOMThreshold` to override the MTOM element size threshold. The MTOM element size threshold is set to a default value of 1000 bytes.

The `Allow MTOM` node property and the `MTOMThreshold` setting can both be overridden in the local environment.

The overrides that apply at a SOAPReply node are:

- `LocalEnvironment.Destination.SOAP.Reply.AllowMTOM`, which can have a value of true or false
- `LocalEnvironment.Destination.SOAP.Reply.MTOMThreshold`, which is an integer value in bytes

The equivalent overrides for a SOAPRequest or SOAPAsyncRequest node are:

- `LocalEnvironment.Destination.SOAP.Request.AllowMTOM`, which can have a value of true or false

- `LocalEnvironment.Destination.SOAP.Request.MTOMThreshold`, which is an integer value in bytes

Configuring message flows to process timeouts
Connect the HTTP Timeout terminal of the HTTPInput or SOAPInput nodes to further nodes to process timeouts.

Before you begin

- Read the message flows overview.
- Read about the options that you have for processing web service messages, and learn more about SOAP and HTTP.

About this task

You can configure message flows that start with an HTTPInput or SOAPInput node by connecting the HTTP Timeout terminal to further nodes for processing timeouts:

- On HTTPInput nodes, messages are propagated through this terminal only when you have configured your integration servers such that the HTTP nodes are using the embedded integration server listener.
- On SOAPInput nodes, messages are propagated through this terminal only when you are using an HTTP binding, and you have configured your integration servers such that the SOAP nodes are using the embedded integration server listener.

If these conditions are not met when you deploy the BAR file for the message flow that includes one of these nodes, a warning is generated, and the path of the message flow that you have connected to the HTTP Timeout terminal is ignored. No further warnings are generated until the next restart.

To set a static timeout value in an input node:

1. Create a message flow, or open an existing flow.
2. In the Message Flow editor, select the input node for this message flow. The node properties are displayed in the Properties view (below the editor pane).
3. Set an appropriate time for the timeout interval in the property `Maximum client wait time`. The default interval is 180 seconds.

   If this time expires, and you have not connected one or more nodes to the HTTP Timeout terminal, the listener that received the client request message responds with a SOAP Fault message indicating that a timeout has occurred.
4. If you want to provide customized timeout processing, connect one or more nodes to the HTTP Timeout terminal. You must include in this sequence the reply node that matches the input node. Therefore, if your message flow starts with an HTTPInput node, you must include an HTTPReply; if your message flow starts with a SOAPInput node, you must include a SOAPReply node.

To set a dynamic timeout value in an input node:

- Override the timeout value set on the input node by using the Java plug-in API to update it in a JavaCompute node using the `MbUtilities.changeIdentifierTimeout()` method. The following code shows an example of the `changeIdentifierTimeout` method:

```
MbMessage localEnv = assembly.getLocalEnvironment();
MbElement rootElem = localEnv.getRootElement();
MbElement repIdElement = rootElem.getFirstElementByPath(
   "/Destination/SOAP/Reply/ReplyIdentifier");
Object repId = repIdElement.getValue();
boolean success = changeIdentifierTimeout((byte[])repId, timeout);
```

- Override the timeout value set on the input node by using the CHANGEIDENTIFIERTIMEOUT function.

You can derive the value that you use to replace the existing value by several means; for example:

- Read a record from a database.

- Use a value from a field within the message body.

By propagating from the HTTP Timeout terminal you can then change the contents of the responses that your message flow sends to the client. The processing on the sequence of nodes that you connect to the HTTP Timeout terminal is also subject to a further timeout, so that the client always gets a response within a known timeout interval.

When a message is propagated from the HTTP Timeout terminal the message tree contains the input headers of the original input message and a message body that is the fault timeout message. The original message body along with other information relating to the timeout can be accessed in the LocalEnvironment message tree. For example, the following record can be found in the LocalEnvironment:

```
(0x01000000:Name):HTTP = (
  (0x01000000:Name):Input = (
    (0x01000000:Name):Timeout = (
      (0x03000000:NameValue):OriginalClientLastWaitTime = 10 (INTEGER)
      (0x03000000:NameValue):OriginalClientWaitTime     = 15 (INTEGER)
      (0x03000000:NameValue):OriginalMessageMadeTheFlow = TRUE (BOOLEAN)
      (0x03000000:NameValue):OriginalRequestIdentifier  =
         X'485454500000000000000000000000c00c000000000000' (BLOB)
      (0x03000000:NameValue):OriginalInboundMessage     = X'3c3e' (BLOB)
    )
  )
)
```

For SOAPInput nodes the SOAPReply node connected on the HTTP Timeout terminal path must send a SOAP fault response message and the reply status code of 500 cannot be altered. For HTTPInput nodes any response message can be sent from the HTTP Timeout terminal and the reply status code can be changed by updating the `LocalEnvironment.Destination.HTTP.ReplyStatusCode` message tree field.

Processing JMS messages

JMS is the standard Java EE messaging API for building enterprise messaging applications. IBM App Connect Enterprise provides built-in input and output nodes for its supported protocols.

The JMS specification offers an abstract, provider-independent messaging interface, so that you can write messaging applications to this common API regardless of the underlying messaging implementation.

JMS offers two common ways of messaging on which to build messaging applications: point-to-point and publish/subscribe (pub/sub). JMS applications send messages to JMS destinations, which can be queues (in the point-to-point domain) or topics (in the pub/sub domain).

JMS is defined in the following documents from Sun Microsystems:

- JMS specifications from Sun Microsystems

JMS can be used as a transport for SOAP as described in the following document issued by the World Wide Web Consortium (W3C):

- SOAP over Java Message Service 1.0 (W3C recommendation)

Java Message Service (JMS) API

IBM App Connect Enterprise supports the Java Message Service (JMS) application programming interface (API).

The JMS is an application programming interface that provides Java language functions for handling messages. Developed by messaging vendors, including IBM, in partnership with Sun Microsystems, Inc., the JMS API provides a common interface to access different enterprise messaging systems, including IBM MQ. This interface is appropriate for point-to-point and publish/subscribe applications.

Messaging clients in JMS are called *JMS clients*, and the messaging system is called the *JMS provider*. A *JMS application* is a business system that comprises JMS clients and at least one JMS provider. Client applications that use the JMS interface are written in the Java programming language, and are therefore restricted to the levels of JVM that are supported on the operating system in the business environment.

If you have existing end-user applications that are written to these interfaces, they can typically run unchanged in an integration node environment. You must create the message flows to interact with these applications from the supported protocols by using the appropriate input and output nodes. IBM App Connect Enterprise provides built-in input and output nodes for its supported protocols, and other nodes that support transformation to and from JMS message formats. You can also create your own user-defined nodes to support additional protocols.

You can also create end-user applications to interact with the integration node.

Using JMS in IBM App Connect Enterprise

Use IBM App Connect Enterprise to connect to applications that send and receive messages that conform to the Java Message Service (JMS) standard.

You can use the built-in JMS nodes in IBM App Connect Enterprise to support the following operations:

- Receive a JMS message as input.
- Receive a JMS message in the middle of a message flow.
- Create a JMS message for output.
- Work with message flows that do not expect JMS messages.

All JMS messages must conform to the Java Message Service Specification version 1.1 or 2.0.

You can create message flows to receive messages from JMS destinations, and send messages to JMS destinations. These destinations are accessible through a connection to a JMS provider. In sending and receiving messages, the JMS nodes behave like JMS clients.

You can also use two transformation nodes, the JMSMQTransform node and the MQJMSTransform node, with the JMSInput, JMSReceive and JMSOutput nodes so that they can interact with nodes that expect a propagated message to contain an MQMD (and MQRFH2) header.

- The JMSMQTransform node takes the output of the JMSInput or JMSReceive node, and produces a message that can be handled by an MQOutput node.
- The MQJMSTransform node transforms a message with an MQMD (and optional MQRFH2) header into a message that is expected by the JMSOutput node.

If you want to change or add content to the headers in JMS messages, you can include a JMSHeader node in the message flow to modify fields without programming.

You can configure the JMS nodes to work with the IBM MQ JMS provider, WebSphere Application Server, and all JMS providers that conform to the Java Message Service Specification version 1.1 or 2.0. IBM App Connect Enterprise supports Java 7(also known as Java 1.7).

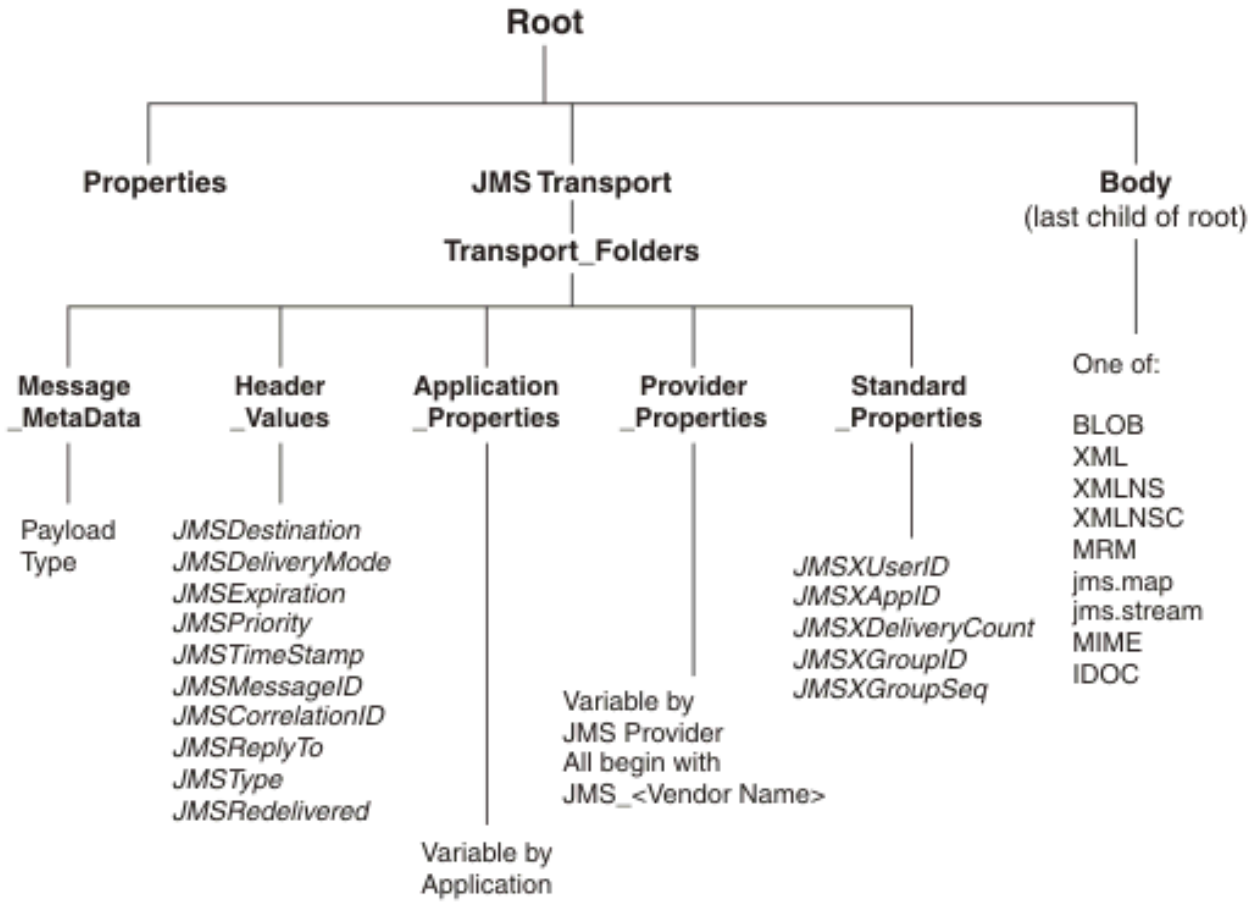
Refer to the following topics for further information about JMS messages and JMS providers:

- “Simplified JMS message representation” on page 852
- “JMS message transformation” on page 853
- “JMS message structure” on page 856
- “JMS message selector” on page 866
- “JMS Transactionality” on page 868
- “JMS message domain properties” on page 869
- “JMS properties for application communication models” on page 869

Simplified JMS message representation

The JMSInput node receives an input message as a Java object, and not as a bit stream wire format (as is the case with an MQInput node). The message does not populate an MQMD and RFH2 header, but instead populates a new message tree that represents a JMS message in a more native way.

To represent a JMS message in a message tree, a new canonical form has been created. This message tree allows for representation of JMS message header data, and message properties. The JMS message tree is in a format that is recognizable to Java programmers.



For details about the structure and content of the JMS message tree, see "Representation of messages in the JMS Transport" on page 858.

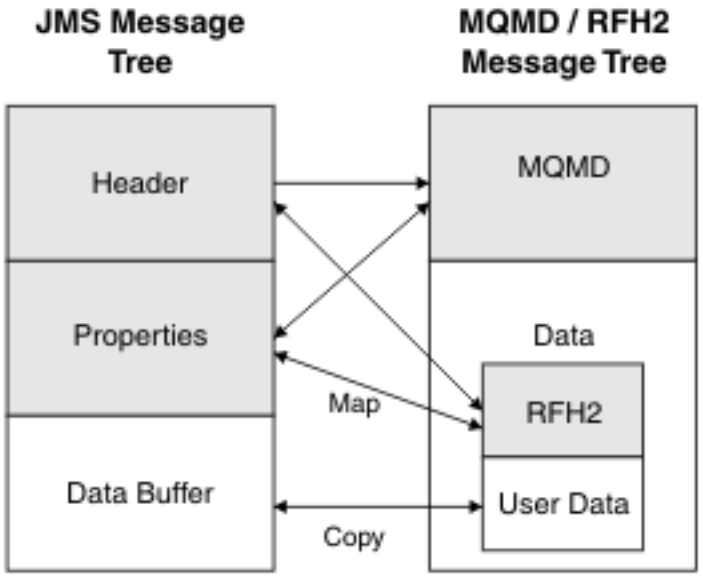
JMS message transformation
The JMSInput and JMSOutput nodes expect JMS messages, and therefore expect a native JMS message tree representation.

You can use the following nodes to transform messages between a WebSphere MQ JMS message tree and a JMS message tree:

- JMSMQTransform node
- MQJMSTransform node

These nodes do not have any configurable properties. The JMSMQTransform node transforms a native JMS message tree to an IBM MQ JMS message tree, and the MQJMSTransform node performs the transformation in the opposite direction.

The following diagram provides an overview of the mapping scheme that is used:



This mapping diagram uses the same scheme as the IBM MQ JMS provider to convert between a JMS message and an MQMD or MQRFH2 message.

When transforming between an IBM MQ message tree and a native JMS message tree, the transformation nodes copy elements from different parts of a message tree:

- The following fields are copied from the JMS message to the MQMD, if they exist in the incoming JMS message:

JMS field	**MQMD field**
JMSMessageID	MsgId
JMSCorrelationID	CorrelId
JMSPriority	Priority
JMSDeliveryMode	Persistence
JMSQApplid	PutApplName
JMSUser	UserIdentifier
JMSXDeliveryCount	BackoutCount - 1
JMSTimeStamp	PutDate, PutTime

- The following fields are copied from the JMS message to the MQRFH2 JMS folder:

JMS field	**MQRFH2 JMS field**
JMSDestination	Dst
JMSDeliveryMode	Dlv
JMSExpiration	Exp
JMSPriority	Pri
JMSTimestamp	Tms
JMSCorrelationID	Cid
JMSReplyTo	Rto

- The following fields are copied from the MQMD to the JMS message:

MQMD field	JMS field
Expiry	JMSExpiration
Persistence	JMSDeliveryMode
Priority	JMSPriority
MsgId	JMSMessageID
CorrelId	JMSCorrelationID
BackoutCount = 0	JMSRedelivered = false
BackoutCount > 0	JMSRedelivered = true
GroupId	JMSGroupid
MsgSeqNumber	JMSGroupseq
UserIdentifier	JMSUser
PutApplName	JMSApplid
PutDate, PutTime	JMSTimeStamp

- The following fields are copied from the MQRFH2 JMS folder to the JMS message:

MQRFH2 JMS field	JMS field
Dst	JMSDestination
Dlv	JMSDeliveryMode
Pri	JMSPriority
Cid	JMSCorrelationID
Rto	JMSReplyTo

Example message flow scenario: JMSInput node to MQOutput node

1. A JMSInput node is configured to subscribe to topic ABC.
2. An application that is connected to the JMS server publishes on topic ABC.
3. A publication is received at the JMSInput node.
4. The node extracts data from the JMS message.
5. The JMS message is passed to the JMSMQTransform node where the message is converted to an IBM MQ message.
6. The MQOutput node receives the IBM MQ message, and publishes the message on an IBM MQ queue.

The final destination is an IBM MQ queue, therefore the message must pass through a JMSMQTransform node to convert the message tree to an IBM MQ JMS format before it reaches the MQOutput node.

Example message flow scenario: MQInput node to JMSOutput node

1. An MQInput node receives a message from an IBM MQ queue.
2. The MQInput node creates an IBM MQ message.
3. The MQ message is passed to the MQJMSTransform node where the message tree is converted to a JMS format.
4. The JMSOutput node receives the JMS message and publishes the JMS message on topic XYZ.

Additional examples

These examples show some of the solutions that you can achieve when you use the JMS Transport. Other solutions are possible; for example, the message can be passed to a Compute node or a JavaCompute node and the contents can be modified as required.

JMS message structure
JMS messages have a defined structure that includes headers and payloads.

The following figure depicts the JMS message structure:

Header

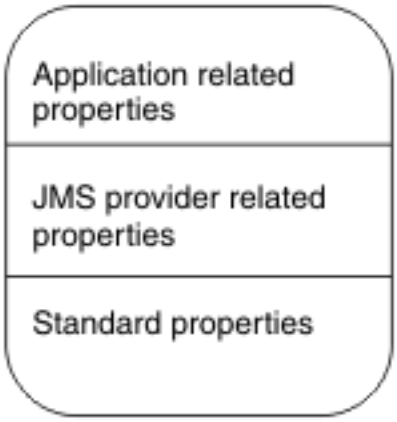
A header must be present in every JMS message, and it is assigned automatically. Most of the values in the header are set by the JMS provider when the message is put to a JMS destination. Some values can be declared by the JMS client when it creates a JMS session, or when it creates the message consumer or

producer; for example, *JMSDeliveryMode*, *JMSExpiration*, *JMSReplyTo*, and *JMSCorrelationID* are created when the JMS client creates a JMS session or creates the message consumer or producer.

The data elements of each header comprise name-value pairs and they can be any of the Java following types: Boolean, byte, short, char, long, int, float, double, string, or byte[].

Properties

The properties are optional, and can be divided into the following subsections:



- *Application-related properties*

  A Java application can assign application-related properties, which are set before the message is delivered. The property names of the application are meaningful only to the sending and receiving applications.
- *Provider-related properties*

  Every JMS provider can define proprietary properties that can be set either by the client or automatically by the provider. Provider-related properties are prefixed with *JMS_* followed by the vendor name and the specific property name. For example, the IBM MQ JMS client sets the provider property to be *JMS_IBM_MsgType*.
- *Standard properties*

  These properties are set by the JMS provider when a message is sent. The JMS provider vendor can choose to not support any standard properties, to support some standard properties, or to support all standard properties. Standard property names start with *JMSX*; for example: *JMSXUserid* or *JMSXDeliveryCount*.

The properties are handled as name-value pairs, and they can be any of the following Java types: Boolean, byte, short, char, long, int, float, double, string, or byte[].

Payload

The payload type defines the JMS message. It can be one of the six JMS message types that are described in "JMS message types" on page 857.

JMS does not define a wire format. The Java Message Service Specification describes the physical representation of how a message is structured.

JMS message types
JMS defines six message interface types; a base message type and five subtypes. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content.

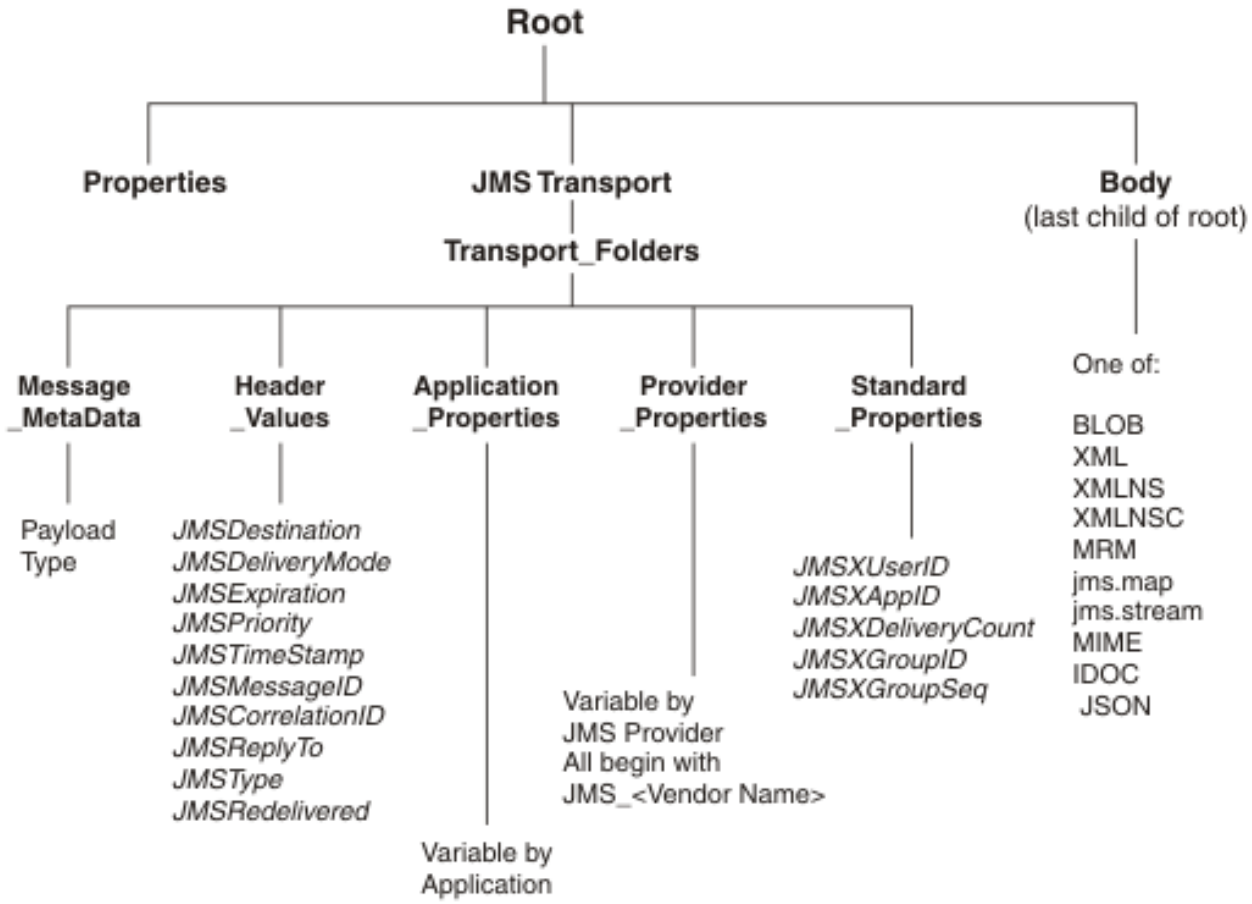
JMS specifies only the interface and does not specify the implementation. This approach allows for vendor-specific implementation and transportation of messages while using a common interface.

The following table describes the six message types:

Message type	Description
Message	The base class. This message type is used for event notification, and does not have a payload.
BytesMessage	The payload is stored as an array of bytes. This message type is useful for exchanging data in a format that is native to the application, and when JMS is used as a transport between two systems, where the JMS client does not know the message payload type. Use this message type to transmit XML messages to ensure that the message is transmitted efficiently, and is not subject to unnecessary data conversion.
TextMessage	Data is stored as a string. This message type is useful for exchanging simple text messages.
StreamMessage	A Stream message is a sequence of primitive Java types. The message object tracks the order and the types of these primitives within the stream. Formal conversion rules apply; for example, an exception is thrown if a JMS application tries to read a double value as a short value. Refer to the Java Message Service Specification version 1.1 or 2.0 for a full list of the conversion rules. `21ABCDEFGH32.345` is an example of a StreamMessage payload. It consists of the following three fields: • An Integer, 21 • A String, ABCDEFGH • A Float, 32.345 If the data structure is unknown, the generic method `readObject()` can be used to return the next object in the stream. If the structure of the data is known, the JMS client can be specific about the type of object being accessed.
MapMessage	The payload of a MapMessage is stored as a set of name-value pairs. The name is defined as a string and the value is typed. The MapMessage is useful for delivering keyed data that can change from one message to the next. `NumberOfCopies:5` is an example of a MapMessage payload, where `NumberOfCopies` is the key and 5 is the value. Data can be accessed by using `getMapNames()`, which returns a Java Enumeration object. It is possible to iterate through the MapMessage by using `hasMoreElements()` to retrieve the mapped name-value pairs.
ObjectMessage	The Object message carries a serializable Java Object as its payload. It is useful for exchanging Java objects.

Representation of messages in the JMS Transport
Messages that are sent in the JMS Transport are represented by the JMS Transport message tree.

The following figure depicts the JMS message tree that is propagated from the JMSInput node and can be propagated to the JMSOutput and JMSReply nodes. The JMSOutput and JMSReply nodes populate details of the JMS message that is sent to the LocalEnvironment WrittenDestination folder, as described in node.



JMSTransport

- *Header_Values* subfolder:

  This subfolder is mandatory and is always created.

- Properties subfolders:

  JMS message properties are optional; if they are present in the message, they are stored in the appropriate properties subfolder.

- *Message_MetaData* subfolder:

  This subfolder is included to preserve the payload type of the JMS message. The folder is used by the JMSOutput node when it creates a JMS message. The payload type can be one of the values shown in the following table.

Message type	Payload values
Base JMS message with no payload	jms_none
TextMessage	jms_text
BytesMessage	jms_bytes
MapMessage	jms_map
StreamMessage	jms_stream
ObjectMessage	jms_object

Body

The message payload is stored in the body folder, which is the last child of Root. The payload is transferred by using one of the following message domain parsers:

- DFDL
- XMLNSC
- JSON
- BLOB
- JMSMap
- JMSStream
- MIME
- XMLNS
- MRM
- IDOC

JMS message as input
The JMS message is a Java object, and therefore you cannot parse the message as a bit stream. When the message is received, the header data, property data, and payload data are extracted by using the JMS API.

For information on the header, properties, and payload of JMS messages, refer to "JMS message structure" on page 856.

The following topics describe how the different parts of the JMS message are obtained, and how the message is parsed:

- "JMS input message header and property data" on page 860
- "JMS Message payload" on page 862
- "JMS message payload and appropriate parser" on page 863

JMS input message header and property data
The JMSInput node obtains header and property data from JMS messages.

Header data

The JMSInput node extracts header data from messages by using JMS API methods. Header data is stored as name-value pairs in the *Header_Values* folder. The API methods return the value; for example, to get the value for the header field *JMSTimestamp*, the JMSInput node uses the `getJMSTimestamp( )` method. A similar method is provided for each of the following fixed header fields:

- JMSDestination
- JMSDeliveryMode
- JMSExpiration
- JMSPriority
- JMSTimeStamp
- JMSMessageID
- JMSCorrelationID
- JMSReplyTo
- JMSType
- JMSRedelivered

Property data

In a similar way to how the header data is obtained, the JMSInput node extracts property data from messages by using JMS API methods. Property data is stored as name-value pairs in the properties folders. The API method returns a value for every property name with which it is supplied.

XML representation of header and property data

The JMSInput node uses the header and property data to create an XML representation of the JMSTransport folders. The node passes the XML data to the JMSTransport parser as a byte array. The byte array is then used to populate or to refresh the elements in the message tree.

Preservation of Java type

A scheme is not required to preserve knowledge of the Java type because the header value Java types are fixed and known. The JMS message properties are optional, therefore a scheme is required to preserve the Java type of the property values. The scheme used is that which is implemented by the IBM MQ JMS client and the Real-timeInput node.

Java type information is represented as a metadata in the form of a keyword `dt='DataType'` where *Datatype* is a string. The Java type is passed in the XML as part of the element name `<ElementName dt='DataType'>Value</ElementName>`. *Datatype* can be any of the following values:

Datatype value	Definition
String	Any sequence of characters, excluding < and &
Boolean	The character `0` or `1`, where `1` is equal to "true"
bin.hex	Hexadecimal digits representing octets
I1	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -128 to 127 inclusive.
I2	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -32768 to 32767 inclusive.
I4	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -2147483648 to 2147483647 inclusive.
I8	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the range -9223372036854775808 to 9223372036854775807 inclusive.
int	A number, expressed by using the digits 0..9, with optional sign (no fractions or exponent). The value must lie in the same range as the datatype value`I8`. This number can be used in place of one of the `I*` types if the sender does not want to associate a particular precision with the property.
R4	A floating point number, expressed by using the digits 0..9, optional sign, optional fractional digits, optional exponent. Magnitude <= 3.40282347E+38, and>= 1.175E-37

Datatype value	Definition
R8	A floating point number, expressed by using the digits 0..9, optional sign, optional fractional digits, optional exponent. Magnitude <= 1.7976931348623E+308, and>= 2.225E-307

JMS Message payload
How payload is extracted from the JMS message for each of the JMS Message types.

The payload for some of the JMS message types can be extracted as a whole from the message object by using the JMS API. The payload is passed as a bit stream to an integration node parser. This is true for the following message types:

- BytesMessage
- TextMessage
- ObjectMessage

  Additional processing is required to deal with the ObjectMessage payload because the JMS ObjectMessage payload is a serialized Java Object.

  The JMSInput node obtains the payload by calling `getObject( )` on the message. `getObject( )` returns a de-serialized object of the original class. This class definition must be made available to the JMSInput node, and you should ensure that it is accessible through the integration node's Java class path. (The class path is defined in the mqsiprofile batch file, which is in the integration node's executable directory; for example, on Windows, this is `mqsiprofile.cmd` in the *`install_dir`*`/bin` directory.) The JMSInput node invokes the BLOB parser, which creates the message body by using a bit stream that is created from the object.

  The Java Object can be subsequently re-serialized in a JavaCompute node or a user-defined extension, and is updated by using its method calls.

The payload for MapMessage and StreamMessage can be extracted only as individual elements and must be reformatted by the JMSInput node before it can be used to create the message body.

- **MapMessage payload**

  The JMSMap domain is a synonym for the integration node XML parser, which expects a stream of XML data. MapMessage payload data however, is extracted as sets of name-value pairs from the message object. The JMS API is used to obtain the name-value pairs.

  The JMSInput node appends each name-value pair to a bit stream as an XML element and value, and preserves the type of the value by using the `dt=` attribute.

  The following example shows the XML that is generated by the JMSInput node for the MapMessage payload:

```
<map>
    <Item_8_of_10_Char dt='char'>A</Item_8_of_10_Char>
     <Item_5_of_10_Double dt='r8'>999999.0</Item_5_of_10_Double>
     <Item_10_of_10_String>Last Map Item</Item_10_of_10_String>
     <Item_9_of_10_Boolean dt='boolean'>0</Item_9_of_10_Boolean>
     <Item_2_of_10_Integrer dt='i4'>999</Item_2_of_10_Integrer>
     <Item_3_of_10_Short dt='i2'>9999</Item_3_of_10_Short>
     <Item_7_of_10_Byte dt='i1'>9</Item_7_of_10_Byte>
     <Item_6_of_10_Float dt='r4'>2.24</Item_6_of_10_Float>
     <Item_1_of_10_String>P2P Map Msg Number:1</Item_1_of_10_String>
     <Item_4_of_10_Long dt='i8'>99999</Item_4_of_10_Long>
```

```
            </map>
```

In this example, the message contains 10 fields. The field names have been generated by a JMS Client application, and take the form `item_n_of_x_t`, where:

- n is the sequence number in which the item was added to the message,
- x is the total number of items in the map,
- `t` is the type of the value.

The map data is not returned from the JMS API the order in which it was received.

If a JMSInput node receives a JMS MapMessage that contains characters that would be invalid in an XML tag, it escapes the names as follows:

1. The XML tag of the name element is formed by truncating the JMS message name element up to the location of the first invalid character.
2. The original name from the JMS name element is preserved in an attribute of the XML tag.
3. The closing XML tag matches the opening tag, without the tag attribute.

For example, a name-value pair of `<ORDER_ITEM[5]><WIDGET>` in a JMS MapMessage would be converted to XML and written to the message tree as `<ORDER_ITEM ibm_original_name="ORDER_ITEM[5]">WIDGET</ORDER_ITEM>`

- **StreamMessage payload**

  The StreamMessage payload data is a sequence of fields, where each field has a specific type. The fields do not have associated names and so a default element name `elt` is used to generate the XML elements. Similar to the MapMessage, the JMS API allows for fields only to be retrieved individually. The JMSInput node extracts fields from the JMS message and appends each to a bit stream in XML format.

  The following is an example of the XML that is generated by the JMSInput node for the StreamMessage payload:

```
<stream>
     <elt>P2P Stream Message  Number :7</elt>
     <elt dt='i4'>999</elt>
     <elt dt='i2'>9999</elt>
     <elt dt='i8'>99999</elt>
     <elt dt='r8'>999999.0</elt>
     <elt dt='r4'>2.24</elt>
     <elt dt='i1'>9</elt>
     <elt dt='char'>A</elt>
     <elt dt='boolean'>0</elt>
     <elt>Last Stream Item</elt>
</stream>
```

  In this example, 10 typed values are added to the StreamMessage by a JMS client application.

JMS message payload and appropriate parser
Configure the JMSInput node properties to specify the message domain that will be used to parse the JMS message payload.

When the JMSInput node creates a message tree from the JMS message payload, the appropriate message domain for that payload must be used. Therefore, the JMSInput node must know the type of JMS message that it expects to receive. The JMSInput node extracts the payload from the JMS message using the appropriate JMS API, then passes the payload data to the parser for the domain. The parser creates the body portion of the message tree.

The message domain is derived according to the following criteria and in the following order of precedence:

1. “The Message domain property is set to a specific domain” on page 864
2. “The Message domain property is left blank (default) and the JMSType header field from the JMS input message is used to indicate the domain” on page 864

3. "The Message Domain property is left blank (default) and the JMSType header field from the JMS input message is also left blank" on page 865

The Message domain property is set to a specific domain

In this case, the node expects to receive only the allowed JMS message types for that domain, as shown by the following table:

Message domain	Valid JMS message types				
	BytesMessage	**TextMessage**	**MapMessage**	**StreamMessage**	**ObjectMessage**
BLOB	•	•			•
XMLNS		•			
XMLNSC		•			
JMSMap			•		
JMSStream				•	
MIME	•	•			
IDOC	•	•			
MRM	•	•			
XML		•			

- If the JMSInput node receives a JMS message type that is not valid for the message domain that is configured in the JMSInput node, the node issues a warning and backs out the message either to the source JMS provider destination, or to the backout destination.
- If you specify the MRM domain, you must also specify the `Message model`, `Message` and `Physical format` node properties.
- If you specify the IDOC domain, you must also specify the `Message model` and `Physical format` node properties.
- If you specify the XMLNSC domain, and you want to validate input messages, you must also specify the `Message model` node property.

The Message domain property is left blank (default) and the JMSType header field from the JMS input message is used to indicate the domain

The JMSType header field must be set according to the URI format shown in the following table. The domain in the `mcd:` string can be uppercase or lowercase.

JMSType	Message domain
`mcd://BLOB`	BLOB
`mcd://MRM/[set]/[type]/[?format=fmt]`	MRM
`mcd://XMLNS`	XMLNS
`mcd://XMLNSC/[set]`	XMLNSC
`mcd://IDOC/[set]/[?format=fmt]`	IDOC
`mcd://MIME`	MIME
`mcd://XML`	XML

- If the JMSInput node receives a JMS message type that is not valid for the message domain configured in the JMSType header, the node issues a warning and backs out the message either to the source JMS provider destination, or to the backout destination.
- If the JMSType field does not conform to this URI format, the message is handled in the BLOB domain.
- For details of the `[type]` syntax, refer to "Specifying namespaces in the Message property" on page 2160 .
- If the XMLNSC domain is specified, use `[set]` only if you want to validate input messages.

The Message Domain property is left blank (default) and the JMSType header field from the JMS input message is also left blank

The message domain is set according to the JMS message Java class as follows:

JMS message type	Message domain
TextMessage	XML
BytesMessage	BLOB
MapMessage	JMSMap
StreamMessage	JMSStream
ObjectMessage	BLOB

JMS message for output
When the JMSOutput node receives a JMS message, it calls the JMSTransport parser to return an XML bit stream containing the JMSTransport section of the message, so that it can be examined and processed.

The node extracts the *Message_MetaData* and obtains the payload type information to identify which JMS message type to create for output. If the *Message_MetaData* folder is not present, the output node creates a BytesMessage by default.

Header data

The JMSOutput node extracts the JMS header data from the XML string, and uses this data to populate the values for the JMS header fields in the message.

Property data

The JMSOutput node extracts the property values from the XML string. The XML elements contain type information that identifies which Java Object type to create for each property value.

Message payload

The message payload is obtained from the JMS message as a bit stream. For TextMessage and BytesMessage payloads, the bit stream can be passed to the JMS API directly to create the appropriate payload.

For MapMessage and StreamMessage payloads, the individual elements must be extracted from the XML bit stream. The output node calls the appropriate JMS API method to create the map or stream fields in the message.

For an ObjectMessage payload, the JMSOutput node reserializes the bit stream payload by using the object class. The object class must be available in the Java class path for the integration node. The class path is defined in the **mqsiprofile** batch file, which is in the directory that contains the executable files for the integration node; for example, on Windows, the file is `mqsiprofile.cmd` in the *install_dir*`/bin` directory.

Sending JMS messages

The JMSOutput node generates and supports:

Sending a datagram message
A message with sufficient information to reach its destination, but without an expectation of there being a response as defined in the node attributes.

Sending a Reply message
The message is treated as a reply as defined by the JMSReplyTo property value.

Sending a Request message
The JMSOutput node sends a message to a defined JMS destination with the expectation of a response from the recipient.

See Using the Message Destination Mode for more information about how you perform these tasks.

Message publication

The message is published to the JMS destination that has been specified as a property of the JMSOutput node. However, if the JMSReplyTo header field is set in the JMS message, the JMSOutput node treats the message as a reply to a previous request, and publishes the message to the JMS destination of the previous request.

JNDI administered objects
JNDI (Java Naming and Directory Interface) is a standard Java extension that provides a uniform API for accessing various directory and naming services.

JMS clients use JNDI to browse a naming service to obtain references to administered objects. *Administered objects* are JMS connection factory and JMS destination objects, where JMS destination objects are topics and queues. Administered objects are created and configured by a system administrator.

To create and configure JNDI administered objects, refer to the JMS provider documentation. If you are using the IBM MQ JMS provider, see the sample JMSAdmin definitions file that is included with IBM MQ and refer to the *Using Java* section in the IBM MQ product documentation online.

Location of JNDI administered objects

JNDI administered objects are stored in the bindings. This storage can be either file system based or based on LDAP (Lightweight Directory Access Protocol). LDAP is a software protocol that enables everyone to locate organizations, individuals, and other resources; for example, locating files and devices in a network, either on the public Internet or on a corporate intranet.

LDAP is part of X.500, which is a standard for directory services in a network.

Naming service

A naming service associates names with distributed objects so that the administered objects are located by using names and not complex network addresses. JNDI provides an abstraction that hides the specifics of the naming service, which makes client applications more portable.

A JMS client specifies a *JNDI InitialContext* to obtain a JNDI connection to the JMS messaging server. The InitialContext is the starting point in any JNDI lookup and acts like the root of a file system. The JMS directory service that is being used determines the properties that are used to create an InitialContext.

JMS message selector
A message selector allows a JMS consumer to be more selective about the messages that it receives from a particular topic or queue.

A message selector uses message properties and headers as criteria in conditional expressions. These expressions use Boolean logic to declare which messages are delivered to a client, such as the JMSInput node.

The following table demonstrates the construction of a message selector. It comprises an identifier, such as the *JMSPriority* header, or an application controlled property *myProperty1*. The selector string must specify an operator followed by a literal.

Element	Valid values
Identifiers	• Property or header field reference (such as *JMSPriority*, *myProperty1*) • The following values are not possible: NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, IS
Operators	AND, OR, LIKE, BETWEEN, =, <>, <,>, <=, >=, IS NULL, IS NOT NULL
Literals	• The two Boolean literals, TRUE and FALSE • Exact number literals that have no decimal point; for example, +25, -399, 40 • Approximate number literals. These literals can use scientific notation or decimal; for example, -21.4E4, 5E2, +34.4928 • String literals, for example `TargetFunctionName='createOrderOnSuccess'`

The JMSInput and JMSReceive nodes provide a free format string *PropertySelector*, to specify selectors that filter or include application properties. These nodes also have properties for specific header properties, where the identifier is implicit and is generated by the node. For the header selectors, the operator and literal part of the string must be specified.

If more than one selector is specified the node generates a composite selector string, where the individual selector strings are concatenated with the AND operator, and each selector string part is wrapped with parentheses.

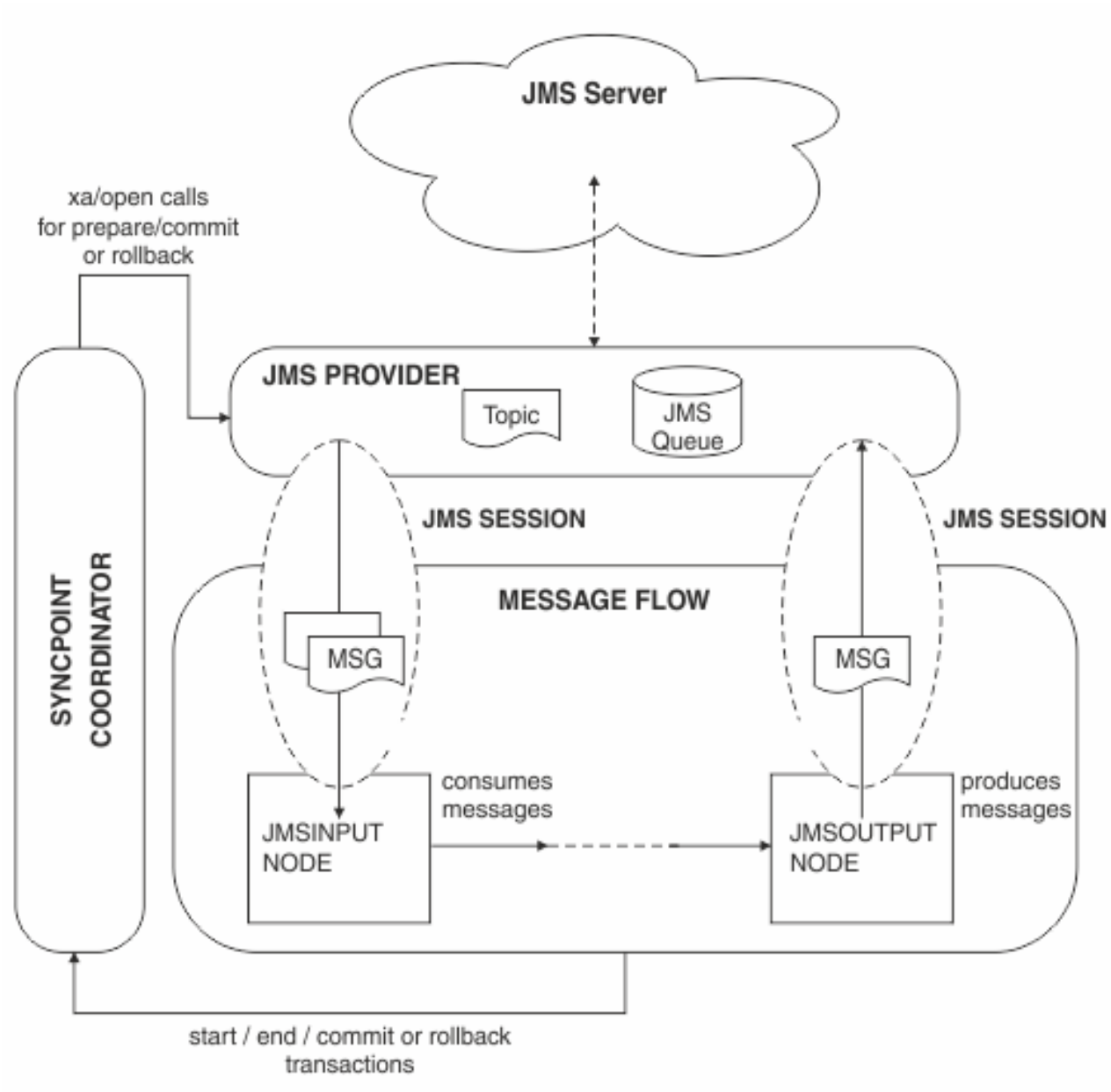
The following are examples for each of the selector properties:

Selector property	Description
PropertySelector	`OrderValue> 100.00` This string is used directly as shown. In the JMSInput node, this property is configured by the **Application property** field.
TimeStamp	`BETWEEN 1057576423231 AND 10575788993265` Messages that are put between these two Java times only (where Java time is milliseconds since 01 Jan 1970) is delivered to the JMSInput node. In this case, the string generated is prefixed with the identifier *JMSTimestamp*.
Delivery Mode	`PERSISTENT` This setting means that only messages marked by the sender as being PERSISTENT should be delivered to the JMSInput node. In this case, the string that is generated is prefixed with the identifier *JMSDeliveryMode*.
Priority	`>= 5 AND <= 8` This setting means that only messages marked by the sender as having a priority 5, 6, 7, or 8 are delivered to the JMSInput node. In this case, the string generated is prefixed with the identifier *JMSPriority*.
Message ID	`> WMBRK123456` This setting returns messages with a Message ID that is greater than the value specified. In this case, the string generated is prefixed with the identifier *JMSMessageID*.

Selector property	Description
Redelivered	FALSE This setting means that messages that have not been redelivered are received by the node. In this case, the string generated is prefixed with the identifier *JMSRedelivered*.
Correlation ID	= WMBRKABCDEFG This setting returns messages whose Correlation ID is equal to the value WMBRKBABCDEFG. In this case, the string generated is prefixed with the identifier *JMSCorrelationID*.

JMS Transactionality
JMS destinations that supply messages to an input node, or receive messages from an output node, can be sync-point coordinated as part of a message flow XA coordinated transaction.

Transactions involving the sync-point coordinator



In this diagram, messages are consumed from a queue by a JMSInput node, and are produced to a JMS queue by a JMSOutput node. The nodes are connected to, and are in session with, a JMS provider. Any message flow input node can inform the external sync-point coordinator when a message flow transaction starts and ends, and whether any resources that have been affected by the flow should be committed or rolled back.

The sync-point coordinator sends XA/Open compliant requests to all participating resource managers to inform them to prepare. Any changes are either committed or rolled back. Resource managers, for example, IBM MQ, DB2 and any XA compliant JMS provider can participate in an XA coordinated transaction.

On distributed systems, IBM MQ is the external sync-point coordinator (transaction manager), which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the flow. On z/OS, Resource Recovery Services (RRS) is the external sync-point coordinator. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

On z/OS, the only JMS provider that is supported is the IBM IBM MQ Java Client, and the only transport mode supported for that client is BIND mode.

Nodes that use JMS transport, such as the JMS and SOAP nodes, can participate in an XA coordinated transaction only if the JMS provider to which they connect supports the XA/Open interface through the JMS XA`Resource` Class. An example JMS provider is the IBM MQ Java Client.

You can specify a generic connection factory (`recover`XAQCF) for recovery of XA coordinated transactions.

In-doubt transactions

In-doubt transactions can occur when a resource manager does not reply to a call from the sync-point manager, where the call is to commit or to rollback resources. During start up of the integration node's IBM MQ queue manager, an initial recovery step is taken to ensure that any in-doubt transactions are resolved before the integration node message flows start to process new input. A JMS provider that participates in integration node global transactions is included in this recovery step.

On operating systems other than z/OS, IBM MQ requires an administration task to be carried out before deployment. This task registers an integration node component, which is a shared library, with the queue manager by referring the shared library to a switch file.

When the integration node's IBM MQ queue manager starts up, it loads the switch file. The switch file forwards XA/Open transaction calls from the sync-point coordinator to the JMS Provider. This ensures that the JMS resources that participate in the transaction can be coordinated in synchronization with other resource managers that are involved in the same transaction.

Additional configuration is required to enable XA coordinated transaction support for the nodes using JMS transport; see “Configuring JMS and SOAP nodes to support globally coordinated transactions” on page 721.

JMS message domain properties
The JMSInput node and the JMSReceive node can receive message payloads that correspond to all of the JMS message types that are specified in the JMS Specification, versions 1.1 and 2.0.

The JMS Specification is defined in Java Message Service Specification. For more information, see “JMS message types” on page 857.

Set the JMSInput node and JMSReceive node properties to specify how the message payload is to be parsed. For more information, see node, node and “JMS message payload and appropriate parser” on page 863.

The JMSOutput node has no message domain properties.

JMS properties for application communication models
JMS clients can operate with both publish/subscribe and point-to-point messages. The publish/subscribe and point-to-point application communication models use virtual channels called *destinations*. In the

publish/subscribe model, the destinations are *topics*. For the point-to-point model, the destinations are known as *queues*.

The following application communication model properties can be configured for JMSInput, JMSReceive and JMSOutput nodes:

Property	Description
Connection Factory Name	A string name that is passed to JNDI to look up the administered connection factory object. The connection factory object is used to create a connection to the JMS destination. • For a client operating as a publish/subscribe client, the connection factory name is for a TopicConnectionFactory. • For a client operating as a point-to-point client, the connection factory name is for a QueueConnectionFactory.
Subscription Topic	The string name that is passed to JNDI to look up the JMS topic destination. The topic is used to create a JMS session when the node is being used to process publish/subscribe messages.
Durable Subscription ID	This is a JMSInput node property only. It is a string identifier that is specified if the node is to subscribe to a durable subscription topic. A durable subscription is one that outlasts the client's connection to a message server. When a durable subscriber is disconnected from the server, the server stores messages that are published. Therefore, when the durable subscriber reconnects, the message server sends all the unexpired messages. Durable subscriptions cannot be unsubscribed from a message flow. A separate administration task unsubscribes a previously registered durable subscription. Some JMS providers supply an administration tool to perform this action.
Source Queue	The string name that is passed to JNDI to look up the JMS queue destination. The queue is used to create a JMS session when the node is being used to process point-to-point messages.

The `Subscription Topic` and `Source Queue` properties are mutually exclusive because they configure the node to work with either the publish/subscribe message model or the point-to-point message model.

The JMSReceive node operates only with point-to-point messages.

A `Durable subscription ID` is not valid without a `Subscriber Topic` property.

Working with JMS

Learn about the concepts and tasks involved in configuring message flows to support JMS messages.

Before you begin

Read the following topics for an overview of the concepts related to processing JMS messages

- "Using JMS in IBM App Connect Enterprise" on page 852
- "Java Message Service (JMS) API" on page 851
- "JMS Transactionality" on page 868
- "JMS message selector" on page 866
- "JMS properties for application communication models" on page 869
- "JMS message domain properties" on page 869

About this task

The following areas are included:

- “Configuring resources for processing JMS messages” on page 874
- “Troubleshooting JMS nodes” on page 885

Connection to different JMS providers
The JMSInput, JMSReceive and JMSOutput nodes are compatible with all JMS providers that conform to the Java Message Service Specification, versions 1.1 and 2.0.

If you want these nodes to participate in coordinated transactions, the JMS provider must support the XAResource interface, as defined in the Java Message Service Specification.

JBoss exception handling

If you use JBoss from a JMSInput node, set the `Handle connection exceptions asynchronously` property to **`True`** on the JMS Providers policy so that the integration node can detect a failure in the JMS connection.

Configuring JMS nodes to communicate with WebSphere Application Server service integration bus
You can configure a stand-alone JMS client and JMS nodes to communicate with service integration bus (SIBus) in WebSphere Application Server Version 6 and Version 7.

About this task

Procedure

1. Complete the following steps in WebSphere Application Server. For more information, see the WebSphere Application Server documentation.
   a) Create a messaging bus.
   b) Add a bus member.
   c) Restart the WebSphere Application Server server.
   d) Create a queue destination on the bus.
   e) Create a JMS queue on the default messaging provider.
   f) Create a Queue Connection Factory (QCF) on the default messaging provider.

      Ensure that the messaging provider URL is specified in the QCF definition, particularly if the JMS client and messaging bus are on different computers. The provider endpoint URL must have the following format:

      ```
      bus_member_host_name:7276:BootstrapBasicMessaging
      ```

      where 7276 is the default SIB endpoint address. Do not use 127.0.0.1 or localhost for the bus member host name.

2. Test the WebSphere Application Server configuration by using a stand-alone JMS client and completing the following steps.
   a) Put the following two JAR files in your class path: `com.ibm.ws.sib.client.thin.jms_7.0.0.jar` and `com.ibm.ws.ejb.thinclient_7.0.0.jar`.

      Copy these JAR files from the WebSphere Application Server Version 7 installation directory under the runtimes subdirectory. If you are using a non-IBM JRE, you also need the `com.ibm.ws.orb_7.0.0.jar` file.
   b) Ensure that the provider URL is set to `iiop://`*`WAS_server_host_name`*`:`*`boot_strap_port`*.
   c) Ensure that you specify the correct boot strap port.
   d) Ensure that the Queue Connection Factory and JMS Queue properties are set to the values defined in the WebSphere Application Server configuration.
   e) Compile the JMS client code.
   f) Run the JMS client with the following IBM ORB debug parameters turned on.

      ```
      java -Dcom.ibm.CORBA.Debug=true -Dcom.ibm.CORBA.CommTrace=true -Dcom.ibm.CORBA.D
       ebug.Output=client.log JMS_Client_Class
      ```

      This command produces CORBA debug output in the `client.log` file in the same directory.
3. Complete the following steps in IBM App Connect Enterprise.
   a) Stop the integration server.
   b) Create a directory (for example, `c:\WebSphere_WAS_Client`) and copy the following two JAR files from WebSphere Application Server Version 7 Thin Client for JMS.
      - `com.ibm.ws.sib.client.thin.jms_7.0.0.jar`
      - `com.ibm.ws.ejb.thinclient_7.0.0.jar`

      Alternatively, you can copy these files from the WebSphere Application Server installation directory *`WAS home`*`/runtimes`.
   c) Configure the JMS service in IBM App Connect Enterprise by using the **mqsichangeproperties** command. The JMS provider WebSphere_WAS_Client exists; therefore you can change the client JAR file path for that provider.

      ```
      mqsichangeproperties integrationNodeName -c JMSProviders -o WebSphere_WAS_Client -n
       jarsURL -v WAS_thin_client_JAR_file_path
      ```
   d) Optional: Verify that your JMS service is configured correctly.
   e) Configure the JMSInput node as shown in the following example.

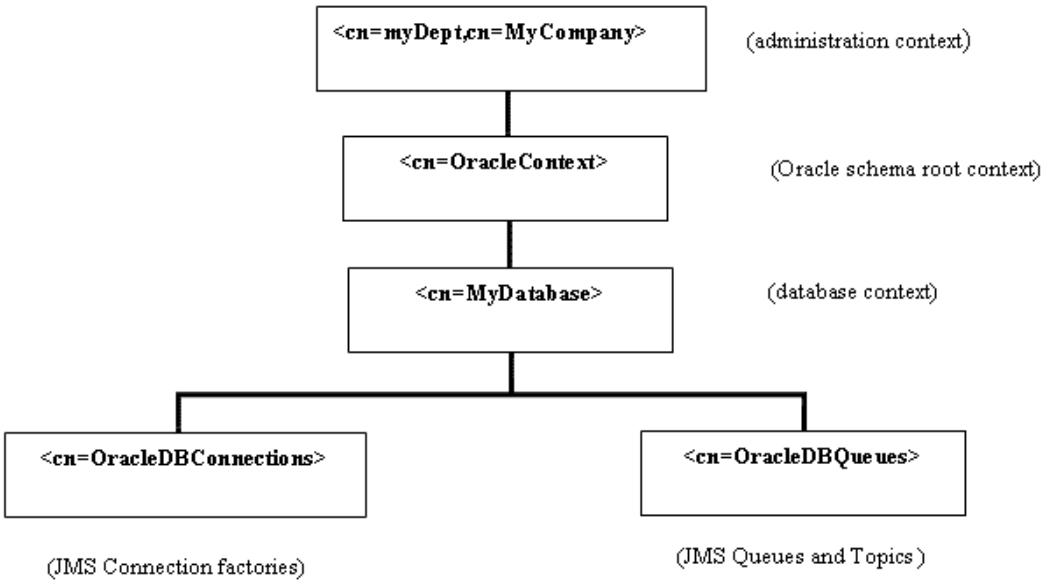
      For more information about these properties, see node.
      - Specify the name of the JMS provider; for example, Client for WebSphere Application Server.
      - Specify the initial context factory; for example, `com.ibm.websphere.naming.WsnInitialContextFactory`.
      - Specify the location of the JNDI bindings in the format `iiop://`*`WAS_server_host_name`*`:`*`WAS_server_boot_strap_port`*.
      - Set the connection factory name to QCF.
   f) Ensure that the JMS connection has been established before the message flow starts by using the Windows Event Viewer.

Configuring JMS nodes to communicate with Oracle AQ
You can configure the JMS nodes to communicate with Oracle AQ (Oracle 11g and above). This communication requires an LDAP 3 compliant server to hold definitions for JNDI lookup by the JMS nodes.

About this task

Procedure

1. Complete the following steps in Oracle AQ, referring to Oracle documentation for the specific details of each step.
   a) You must install the Oracle Internet Directory Server (OID) to host the JNDI administered objects for Oracle AQ.
      - You can configure Oracle AQ to register JMS connection factories and destinations (queues and topics) automatically with the OID server when these JMS connection factories and destinations are created in the Oracle database.
   b) Create the database tables to hold the JMS queues and topics.
   c) Create the JMS queues and topics and associate them with the tables created in step 1b.
2. Add definitions for the JMS connections to the LDAP server to permit the integration node JMS nodes to complete JND lookup and connect to the Oracle AQ server.
   a) Register JMS connection factories with the OID LDAP server by using the administrative tools provided by Oracle.
   b) The following diagram describes the shape of the directory tree for JNDI administered objects for Oracle



   AQ.
3. Copy the Oracle AQ JMS client JAR files to a local directory that is accessible by the integration node.
   - The `aqapi.jar` file is found on the Oracle AQ server in directory *`oracle_install_path`*`/rdbms/jlib`.
   - The `ojdbc5.jar` file is found on the Oracle AQ server in directory *`oracle_install_path`*`/jdbc/lib`.
   - The orai18n.jar found on the Oracle AQ server in directory *`oracle_install_path`*`/jlib`.
4. Modify the JMS Providers policy for Oracle AQ to identify the location of the Oracle JAR files.

5. Configure the properties on the JMS Connection tab of the JMS node (input, output, or reply node) as shown in the following example. For more information about these properties, see node.
   - Set the `JMS provider name` property to `Oracle_AQ`.
   - Set the `Initial context factory` property; for example:

     ```
     com.sun.jndi.ldap.LdapCtxFactory
     ```

     For all nodes that refer to the JMS Providers policy, if this property is set on the policy, it overrides the property that is set on the node.
   - Set the `Location JNDI bindings` property; for example:

     ```
     ldap://LDAP_server_address:LDAP_listener_port
     ```

     For all nodes that refer to the JMS Providers policy, if this property is set on the policy, it overrides the property that is set on the node.
   - Set the `Connection factory name` property. This name must be the fully qualified path in the LDAP directory; for example:

     ```
     cn=QCF,cn=oracledbconnections,cn=ORCL,cn=OracleContext,
     ou=MyDept,o=MyCompany
     ```

     Where
     - cn=QCF is the JMS connection factory name.
     - cn=oracledbconnections is the branch for JMS connection factory definitions.
     - cn=ORCL is the Oracle AQ database name.
     - cn=OracleContext is the root of the Oracle RDBMS schema.
     - ou=MyDept, o=MyCompany is the installation-specific LDAP administrative context.
6. On the Basic tab, configure the JMS destinations (queue or topic) properties.
   - Set the `Source queue` property on the JMSInput node. This queue must be the fully qualified path in the LDAP directory; for example:

     ```
     cn=JMS.Queue,cn=oracleDBQueues,cn=ORCL,cn=OracleContext,
     ou=MyDept,o=MyCompany
     ```

     Where
     - cn=JMS.Queue is the JMS queue.
     - cn=oracleDBQueues is the branch for JMS queues and topic definitions.
     - cn=ORCL is the Oracle AQ database name.
     - cn=OracleContext, is the root of the Oracle RDBMS schema.
     - ou=MyDept, o=MyCompany is the installation-specific LDAP administrative context.
7. Before the message flow starts, ensure that the JMS connection has been established by using the Windows Event Viewer.

Configuring resources for processing JMS messages
A number of nodes are provided in IBM App Connect Enterprise for processing and routing JMS messages. Follow the links in this topic to find out how to configure the JMS nodes and other resources for processing JMS messages.

About this task

IBM App Connect Enterprise provides the following nodes for working with JMS messages:

JMSInput node
Use a JMSInput node if the messages are received from a JMS application.

JMSOutput node
Use a JMSOutput node if the messages are sent to a JMS destination.

JMSReceive node
Use a JMSReceive node to receive JMS messages in the middle of a message flow. The JMSReceive node has a similar function to the JMSInput node, but the JMSReceive node can consume or browse JMS queues in the middle of a message flow. The result message is combined with the input message and propagated to the out terminal.

JMSReply node
The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message, and when you have no other routing requirements.

JMSMQTransform node
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a tree structure that is compatible with the format of messages that are produced by the IBM MQ JMS provider.

The JMSMQTransform node can be used to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

MQJMSTransform node
Use the MQJMSTransform node to receive messages that have an IBM MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

You can use the MQJMSTransform node to send messages to existing message flows and to interoperate with IBM MQ JMS and IBM MQ Publish/Subscribe.

JMSHeader node
Use a JMSHeader node to change JMS Header_Values properties, or add, modify, or delete JMS Application properties without programming.

SOAPInput node
Use a SOAPInput node for SOAP messages received from a JMS application.

SOAPReply node
The SOAPReply node sends SOAP messages using JMS transport only to the reply destination specified in the received message. Use the SOAPReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message.

SOAPRequest node
Use a SOAPRequest node to send a SOAP request to a remote web service. This node is a synchronous request and response node, and blocks after sending the request until the response is received.

SOAPAsyncRequest and SOAPAsyncResponse nodes
Use a SOAPAsyncRequest node with a SOAPAsyncResponse node to construct a pair of message flows that call a web service asynchronously.

To use JMS nodes in your message flows, there are additional configuration steps that you might need to complete. See the following topics for information about additional configuration tasks that you might need to complete.

Procedure

- “Windows systems: modifying the queue manager authorization” on page 882
- “Securing JMS connections and JNDI lookups” on page 882
- “Enabling a JMS provider's proprietary API” on page 227
- “Processing bytes messages with JMS nodes” on page 884
- “Configuring JMS and SOAP nodes to support globally coordinated transactions” on page 721.

Configuring JMS and SOAP nodes to support globally coordinated transactions
To include message flow nodes that use JMS transport, such as the JMS and SOAP nodes, in globally coordinated transactions, you must complete additional configuration.

Before you begin

Review "Configuring JMS and SOAP nodes for local transactions" on page 700 to understand what properties you must set on a SOAP or JMS node to use globally coordinated transactions.

Note: Global (XA) transaction coordination for nodes that use JMS transport is supported only if the message flow that contains the nodes is deployed to an integration node that has a dedicated queue manager; the integration node cannot share a queue manager with other integration nodes.

About this task

If you require global transaction coordination, choose a JMS provider that conforms to the Java Message Service Specification version 1.1 or 2.0, and that supports the JMS XAResource API through the JMS session.

If you specify your own JMS provider by using the JMS Providers policy, set the `XA is supported` property of the policy to true to indicate that the selected JMS provider supports XA coordinated transactions. If you set this property to true, and the selected JMS provider does not support XA transactions, an exception is raised. If you set this property to false, but the `Transaction mode` property on the node is set to `Yes` and the `Coordinated Transaction` message flow property is selected, an exception is raised.

If the message designer specified a non-XA-compliant provider, only the non-transactional mode is supported. In this case, you must set the `Transaction mode` property to `None` for all JMS and SOAP nodes that use JMS transport.

On distributed systems, an IBM MQ queue manager provides the coordinated transaction support, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing the messages in the flow. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

Procedure

To configure the message flow nodes, complete the following steps:

1. In the IBM App Connect Enterprise Toolkit, switch to the Integration Development perspective.
2. In the BAR file properties, set the `Coordinated Transaction` message flow property value to **yes**.
3. In the Message Flow editor, set the **Transaction mode** property to `Yes` for each node that uses JMS transport that is required in the XA coordinated transaction.
4. Create a Queue Connection Factory, and use either the default name, *recoverXAQCF*, or supply your own name.

   For more information about creating JNDI administered objects, see "JNDI administered objects" on page 866.

5. There are two modes for enlisting an XAResource with the transaction coordinator:
   a. Static enlistment, which means that the XAResource is called to perform a start transaction when any message flow in the same Integration Bus starts a global coordinated transaction.
   b. Dynamic enlistment where the JMS node registers an interest in joining a global transaction when it has work to perform.

   Before deployment on distributed systems, you must set up a stanza for each JMS provider that you want to use. For Windows, the JMS provider switch file is called `DynJMSSwitch.dll`. For all other operating systems, the JMS provider switch file is called `libDynJMSSwitch.so`.

   Follow the steps in the appropriate topic for the operating system, or systems, that your enterprise uses:

   - Linux Linux and UNIX systems
   - Windows Windows systems

     On Windows only, you must also modify the queue manager authorization, as described in "Windows systems: modifying the queue manager authorization" on page 882.

What to do next

The JMS provider might supply additional JAR files that are required for transactional support; for more information, see the documentation that is supplied with the JMS provider. For example, on distributed systems, the IBM MQ JMS provider supplies an extra JAR file, `com.ibm.mqetclient.jar`.

You must add any additional JAR files to the `shared_classes` directory:

- Linux On Linux and UNIX: `var/mqsi/shared-classes`.
- Windows On Windows: `C:\ProgramData\IBM\MQSI\shared-classes`.

For more information, see the section about making the JMS provider client available to the JMS nodes in node.

Configuring the queue manager to coordinate JMS resources on Linux and UNIX

Configure the global (XA) resource managers for the queue manager by defining a stanza in the integration node's queue manager `qm.ini` file for each new JMS provider.

About this task

The JMS provider can be specified by a JMS or SOAP node that is included in a message flow that is running on the integration node.

Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use XA coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.

To configure the queue manager for Linux and UNIX:

Procedure

1. Add a stanza to the queue manager `.ini` file for each JMS provider, in the following format:

```
XAResourceManager:
    Name=<Name>
    SwitchFile=/install_dir/server/lib/libDynJMSSwitch.so
    XAOpenString=<Initial Context Factory>,
              <location of JNDI bindings>'
              <LDAP Principal>,
              <LDAP Credentials>,
              <Recovery Connection Factory Name>,
              <JMS Principal>,
              <JMS Credentials>
```

```
                ThreadOfControl=THREAD
```

The parameters for the stanza are as follows:

<Name>
Required. The name of the defined JMS provider resource manager.

<SwitchFile>
Required. The file system path to the Switch library that is supplied in the `bin` directory of the integration node. There are two options:

a. The library called JMSSwitch, which manages XAResources that statically enlist with the transaction coordinator.

b. The library called DynJMSSwitch, which manages dynamically enlisted XAresources.

install_dir
The location of the IBM App Connect Enterprise installation. This value is required where the LDAP parameters are omitted, but a user-defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
Required. The Initial Context Factory identifier for the JMS provider. This value is set in the JMSInput node property `Initial context factory`.

<Location of JNDI bindings>
Required. Use either the filepath to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply a file path to the bindings file, do not include the file name. For more information about creating the JNDI administered objections, see node or node.

The values for the *Initial Context factory* and *Location of JNDI bindings* in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

<LDAP Principal>
Optional. Specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects. `LDAP Principal` must match the value that is set for the integration node by using the **`mqsicreatebroker`** command. `LDAP Principal` is not currently supported, it is reserved for future use.

<LDAP Credentials>
Optional. Specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects. `LDAP Credentials` must match the value that is set for the integration node by using the **`mqsicreatebroker`** command. `LDAP Credentials` is not currently supported, it is reserved for future use.

<Recovery Connection Factory Name>
Optional. Specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes. If a value is not specified, you must add a default value for `recover`XAQCF to the bindings file. In either case, the Recovery Connection Factory must be defined as an XA Queue Connection Factory for the JMS provider that is associated with the `Initial context factory`.

<JMS Principal>
Optional. Provide the user ID required to connect to a JMS provider, by using a secure JMS Connection Factory. `JMS Principal` is not currently supported, it is reserved for future use.

<JMS Credentials>
Optional. Provide the password that is required to connect to the same JMS provider in conjunction with the JMS principal. `JMS Credentials` is not currently supported, it is reserved for future use.

The JMS Principal and JMS Credentials must be configured together.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma, as seen in the following example stanza.

The example stanza describes IBM MQ Java as the JMS provider for global transactions:

```
XAResourceManager:
```

```
    Name=XAJMS_PROVIDER1
    SwitchFile=/install_dir/server/lib/libDynJMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

- XAJMS_PROVIDER1 is the user-defined name for the resource manager
- /*install_dir*/server/lib/libDynJMSSwitch.so is the installation path
- com.sun.jndi.fscontext.RefFSContextFactory is the *<Initial Context Factory>*
- /Bindings/JMSProvider1_Bindings_Directory is the location of the bindings file
- myJMSuser1 is the *<JMS Principal>*
- passwd is the password used in *<JMS Credentials>*

In this example, the optional fields <LDAP Principal>, <LDAP Credentials>, and <Recovery Connection Factory Name> are not required, therefore the positional comma delimiters only are configured in the XAOpenString stanza.

2. Update the Java PATH environment variable for the queue manager of the integration node to point to the bin directory in which the switch file is located.
   For example:

```
install_dir/server/bin
```

Switch files are installed in the *install_dir*/server/lib directory. To simplify the contents of the qm.ini file, create a symbolic link to the switch file for the queue manager to retrieve from the /var/mqm/exits64 directory. For example:

```
ln -s install_dir/server/lib/libDynJMSSwitch.so /var/mqm/exits64/libDynJMSSwitch
```

What to do next

Check your configuration:

- In the message flow, ensure that the coordinated property is enabled by using the IBM App Connect Enterprise Archive editor.
- Ensure that each message flow node that is part of the XA transaction is set to the global transaction mode.
- Ensure that the service ID that is used for the integration node and the queue manager is the same user ID.
- Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
  - If you create the bindings by using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.

For more information, see Transactional support in the IBM MQ product documentation online.

Configuring the queue manager to coordinate JMS resources on Windows
Use IBM MQ Explorer to configure the global (XA) resource managers for the queue manager.

About this task

Before you deploy a message flow in which the `Transaction mode` property is set to `Global` or `Yes`, and is intended to use XA coordinated transactions, modify the queue manager `.ini` file to include extra definitions for each JMS provider resource manager that participates in globally coordinated transactions.

To configure the queue manager for Windows:

Procedure

1. Open IBM MQ Explorer.
2. Select the queue manager for your integration node and click **Properties**.
3. Select **XA resource managers** in the left pane and click **Add**.
4. Complete the fields to define a new resource manager:
   a) **Name**: Enter the name of the resource manager; for example, `IIBWMQJMS`.
   b) **SwitchFile**: Enter **DynJMSSwitch**.
   c) On **XAOpenString**, enter the following values, which are comma delimited and positional. Represent missing optional parameters by a comma if you include other parameters later in the string.

   Initial Context Factory
   The Initial Context Factory identifier for the JMS provider; this value is required.

   Location of JNDI bindings
   Either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. If you supply the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details about creating the JNDI administered objects; this value is required.

   The values for the *Initial Context factory* and *Location of JNDI bindings* in the stanza must match the values that you specified in the JMS or SOAP nodes in the message flows.

   LDAP Principal
   Optional: The principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects. `LDAP Principal` is not currently supported, it is reserved for future use.

   LDAP Credentials
   Optional: The credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

   All LDAP parameters must match the values that you specified on the **`mqsicreatebroker`** command. `LDAP Credentials` is not currently supported, it is reserved for future use.

   Recovery Connection Factory Name
   Optional: The name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non-default name is required.

   The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If you do not specify a name, a default factory that is called

recoverXAQCF is used. In either case, this value must refer to a JNDI administered object that has already been created.

JMS Principal
The user ID that is required to connect to a JMS provider, using a secure JMS Connection Factory. `JMS Principal` is not currently supported, it is reserved for future use.

JMS Credentials
The password that is required to connect to the same JMS provider with the JMS principal. `JMS Credentials` is not currently supported, it is reserved for future use.

The JMS Principal and JMS Credentials must be configured together.

a) **XACloseString**: Leave this field blank.

b) **ThreadOfControl**: Set the value to `Thread`.

5. Click **OK** to complete the XA resource manager definition.
6. Click **OK** to close the queue manager properties dialog.
7. Click **File** > **Exit** to close IBM MQ Explorer.
8. Copy the `DynJMSSwitch.dll` file to the `\exits64` folder in the IBM MQ installation directory.
9. Copy the `DynJMSSwitch32.dll` file to the `\exits` directory in the IBM MQ installation directory and then rename the file to `DynJMSSwitch.dll`.
10. Set the `XAOpenString` property to a string value as follows: *`Initial Context,location JNDI,Optional_parms`*.
11. Set the `ThreadOfControl` property to `Thread`.

What to do next

- Check your configuration:
  - In the message flow, ensure that the coordinated property is enabled by using the IBM App Connect Enterprise Archive editor.
  - Ensure that each node that must be part of the XA transaction is set to the global transaction mode.
  - Ensure that the service ID that is used for the integration node and the queue manager is the same user ID.
  - Ensure that the JNDI connection factory objects that the JMS nodes use for a global transaction are configured to be of the type MQXAConnectionFactory, MQXAQueueConnectionFactory, or MQXATopicConnectionFactory.
    - If you create the bindings by using JMSAdmin, use the command **DEF XAQCF** or **DEF XATCF**, instead of **DEF QCF** or **DEF TCF**, when you define your connection factory.
- Authorize the integration node and queue manager to access shared resources that are associated with the JMS provider; see "Windows systems: modifying the queue manager authorization" on page 882.

For more information, see Transactional support in the IBM MQ product documentation online.

Troubleshooting

IBM App Connect Enterprise examines the IBM MQ registry to determine if the queue manager is XA enabled. If IBM App Connect Enterprise is unable to locate the location of the queue managers working directory, an error is issued at startup. You can specify this location by setting the following environment variable in the `mqsiprofile` script: `MQ_DATA_PATH` For example:

```
MQ_DATA_PATH="C:\Program Files (x86)\IBM\IBM MQ"
```

Windows systems: modifying the queue manager authorization
Authorize the integration server and queue manager to access shared resources that are associated with the JMS provider.

Before you begin

Set up the JMS Providers policy (see Making the JMS provider client available to the JMS nodes (in the JMSInput node topic) or Making the JMS provider client available to the JMS nodes (in the JMSOutput node topic).

About this task

If you specified an existing queue manager when you created the integration node, check that its administrative ID is the same ID as that used for the service ID of the integration node. If the ID is not the same, change the queue manager ID to be the same as the integration node service ID.

Complete the following steps on the Windows system on which the integration node is running:

Procedure

1. Click **Start** > **Run** and enter dcomcnfg. The Component Services window opens.
2. In the left pane, under Console Root, expand **Component Services** > **Computers** > **My Computer** and click **DCOM Config**.
3. In the right pane, under DCOM Config, right-click the IBM MQ service labelled **IBM MQSeries Services**, and click **Properties**.
4. Click the **Identity** tab.
5. Select **This user** and enter the user ID and password for the integration node service ID to associate that ID with the queue manager.
6. Click **OK** to confirm the change.

Securing JMS connections and JNDI lookups
If you want additional security for JMS connectivity and the JMS nodes or SOAP nodes using JMS transport, two configuration options are supported.

About this task

When you include JMS nodes in your message flow, you can optionally secure JMS connection resources. You can secure one, both, or neither of these options, depending on the level of security and access that you want to enforce.

Procedure

1. To secure a JMS connection:
   a) Specify the Connection Factory Name property on the node.
      You must set this property for every node using JMS transport.
   b) Use the **mqsisetdbparms** command to authorize the user ID and password for the specified connection factory.
      For example:

```
mqsisetdbparms -w c:\workdir\ACEServ1 -n jms::tcf1 -u myuserid -p secret
```

      where `tcf1` is the name of the connection factory that matches the node property that you set.

2. To secure JNDI bindings lookups:
   a) Specify the Initial Context Factory property on the node.
      You must set this property for every node using JMS transport.
   b) Use the **mqsisetdbparms** command to authorize the user ID and password for the specified context factory.
      For example:

```
mqsisetdbparms -w c:\workdir\ACEServ1 -n jndi::com.sun.jndi.fscontext.RefFSContextFactory
     -u myuserid -p secret
```

      where `com.sun.jndi.fscontext.RefFSContextFactory` is the name of the initial context factory that you set.

Enabling a JMS provider's proprietary API
Some JMS providers provide an alternative interface to the standard JMS specification for particular JMS API calls. In these cases, IBM supplies a Java class to interface with that proprietary API.

About this task

For example, BEA WebLogic uses a component called a *Client Interposed Transaction Manager* to allow a JMS client to obtain a reference to the XAResource that is associated with a user transaction.

If the WebSphere IBM Integration JMS nodes use BEA WebLogic as the JMS provider, and the nodes must participate in a globally coordinated message flow, you must modify the policy properties that are associated with that vendor (see JMS Providers policy (JMSProviders)). The following table shows the properties that have been added to the policy for BEA WebLogic.

JMS provider	Property	Purpose	Default value
BEA_WebLogic	`Proprietary API handler`	The name of the IBM supplied Java class to interface with a JMS provider's proprietary API.	`com.ibm.broker.apihandler.BEAWebLogicAPIHandler`
	`Proprietary API attribute 1`	The Initial Context Factory class name for the vendor	`weblogic.jndi.WLInitialContextFactory`
	`Proprietary API attribute 2`	The URL of the WebLogic bindings	*URL JNDI bindings*
	`Proprietary API attribute 3`	The DNS name of the JMS server	*Server name*

In the list of JMS provider policy templates, the name of the IBM supplied Java class is set to the default value for the `Proprietary API handler` property. Typically, you do not need to change this value, unless you are instructed to do so by an IBM Service team representative.

Procedure

- Use the Policy editor to modify values of the properties for this JMS provider.
- If you have defined a JMS provider policy, set the value for the `Proprietary API handler` property manually.

Enabling batch acknowledgment for JMS messages
Configure JMS message flows to send a batch acknowledgment for receipt of non-transactional JMS messages.

About this task

You can configure message flows that use the JMS transport to send an acknowledgment to the JMS server of all non-transactional JMS messages that have not previously been acknowledged. You might want to enable batch acknowledgment if there is high network latency and want to send an acknowledgment of received messages only after a threshold has been reached.

JMS flows send acknowledgment responses in accordance with the acknowledge setting on the JMS Session created by the integration server. The default setting is `AUTO_ACKNOWLEDGE`, which causes JMS flows to send an acknowledgment response after receiving each non-transactional message. You can change this setting to use `CLIENT_ACKNOWLEDGE` instead by setting the properties `Client acknowledgment batch size` and `Client acknowledgment batch interval` in the JMS Providers policy. If `CLIENT_ACKNOWLEDGE` is used, the integration server calls the acknowledge() method only after a set time interval has passed, or a set number of messages are received.

Processing bytes messages with JMS nodes
The default behavior of IBM App Connect Enterprise when processing bytes messages can affect clients that are designed to use the readUTF() and writeUTF() methods. Construct an equivalent UTF bit stream by using a Compute node.

About this task

By default, IBM App Connect Enterprise processes bytes messages by using the readBytes() and writeBytes() JMS methods. By using these methods, the payload is written or read as a raw byte array. For the input message, the behavior is based on the serialization of the message tree; for the output message, the resulting bit stream is passed to the user-specified parser to construct a logical tree.

This behavior can affect clients that are designed to use the readUTF() and writeUTF() methods. A UTF string contains encoded length information as well as the raw bit stream. To construct an equivalent UTF bit stream that can be read by the readUTF() method, complete the following steps.

Procedure

1. Add a Compute node immediately before a JMSOutput node.
2. Double-click the Compute node to open the corresponding ESQL file.
3. Use the ESQL shown in the following example to construct an equivalent UTF bit stream from an XMLNSC input message. This bit stream can be understood by a client that uses the readUTF() message.

```
CALL CopyMessageHeaders();

DECLARE byteData BLOB ASBITSTREAM (InputRoot.XMLNSC ccsid
InputProperties.CodedCharSetId);
DECLARE stringData CHARACTER CAST(byteData AS CHARACTER ccsid
InputProperties.CodedCharSetId );
DECLARE dataLen INTEGER LENGTH (byteData);

DECLARE blobLen BLOB CAST(dataLen AS BLOB ENCODING
InputProperties.Encoding);
DECLARE str2byteBlobLen CHARACTER SUBSTRING (CAST(blobLen AS
CHARACTER) FROM 15 FOR 4);

SET OutputRoot.BLOB.BLOB = CAST(str2byteBlobLen as BLOB) ||
byteData ;
```

Troubleshooting JMS nodes
Review possible problems with nodes using JMS transport.

Use Activity log as the first step in diagnosing a problem when something unexpected happens in a JMS message flow. Activity log shows recent activities in your message flows and associated external resources, and can show you at a high level any problems with your JMS resources. You can also view the event log for information about errors that occur.

The following errors might occur:

- "Managing badly formed messages" on page 885
- "Diagnosing problems when using XA coordinated transactions" on page 885
- "Problems with JNDI Administered Objects" on page 886

In all cases of error, if the underlying cause is a JMS exception that has been thrown by the JMS provider, the integration node BIP event message includes the text message from the JMS exception to help diagnosis.

Managing badly formed messages

If a message cannot be processed by the JMS input node, or has been rolled back as part of an XA coordinated transaction, the message is backed out to the source destination. The message is then delivered again to the input node.

To prevent badly formed messages from interrupting the processing of valid messages, the node properties can be configured as described in the following table.

Property	Description
`Backout destination`	This property specifies a JMS destination to which backed out messages are routed if the JMS message property *JMSX_DeliveryCount*, which is set by the JMS provider, exceeds the backout threshold. The JMS destination must be applicable to the message model being used by the node; for example, if a subscription topic has been configured on the node, the JMS destination must also be a topic.
`Backout threshold`	This property specifies the integer value that controls a message that is sent to the backout destination. A threshold value of 3 indicates that if the input node receives a message where the value of the *JMSX_DeliveryCount* property exceeds 3, the message is sent to the backout destination and is removed from the source destination. See node.

Diagnosing problems when using XA coordinated transactions

This problem is not applicable to z/OS.

In addition to the integration node service trace, another trace log is provided to diagnose problems that could occur when a node using JMS transport participates in an XA coordinated message flow transaction. That is, at least one JMS node in the message flow has the `Transaction Mode` property set to `Yes`, and the message flow property `Coordinated Transaction` set to `yes`.

To capture the trace log, complete the following steps:

1. Define an environment variable called *XAJMS_TRACEFILE* that is available to the integration node queue manager.
2. Set the value of the environment variable. This value must be a character string that represents the location and file name of the trace log. For example, on Windows, the variable can be configured as shown in the following example:

```
XAJMS_TRACEFILE = c:\JMSSwitchLog
```

3. When the integration node queue manager starts, it performs a recovery step to resolve any previous integration node transactions that the JMS provider considers to be in doubt. This queue manager process writes to two trace logs during this stage. The two trace logs are:
   - *XAJMS_TRACEFILE valuePID*.txt, where *PID* is the process ID of the queue manager start process. This file is produced from the integration node JMSSwitch library; for more information, see "JMS Transactionality" on page 868.

     The previous example produces a file called JMSSwitchLog2596.txt, where the queue manager start up process ID is 2596.
   - *XAJMS_TRACEFILEXARecoveryTrace*.txt, which is produced by the recovery component of the integration node that connects to the JMS provider.
4. After the integration node queue manager has completed recovery, the integration node starts and creates a file called *XAJMS_TRACEFILE valuePID*.txt, where *PID* is the process ID of the queue manager start process. This file is produced from the integration node JMSSwitch library; for more information, see "JMS Transactionality" on page 868.

Neither of these trace files require extra formatting.

Problems with JNDI Administered Objects

Description of problem: The JMS node is unable to obtain the Initial Context Factory or a JNDI administered object, such as the Connection Factory or JMS destination, and message BIP4640 is issued.

Corrective action

1. Verify that the JNDI bindings have been built correctly, and can be reached at the location specified in the node.
2. Check that the values specified in the node for the Connection Factory Name and Source Queue or Destination Queue properties exist in the JNDI bindings.
3. Ensure that the correct keyword is used to match the location of the bindings:
   - file:// when the administered objects have been created in a .bindings file
   - ldap:// when the administered objects exist in an LDAP directory
   - iiop:// when CORBA is used to access the administered objects
4. When the bindings are file based, do not specify the .bindings file name in the node property.
5. Ensure that the Initial Context Factory Class name is set correctly, as specified in the documentation for the JMS provider.
6. Ensure that the Initial Context Factory Class name does not include a file path.
7. Ensure that a JMS destination (Topic or Source Queue or Destination Queue) specified in the node property exists in the JNDI administered objects.
8. Ensure that the JMS Provider jarsURL property has been set correctly by using the **mqsichangeproperties** command. To verify the value, use the following command:

```
mqsireportproperties INODE -c JMSProviders -o JMSProvider -r
```

The JMS nodes continue to attempt to obtain the JNDI administered objects. Correct any problems and rebuild the bindings. The JMS node should automatically detect the changes and attempt to start.

- If the problem is resolved by rebuilding the bindings, the JMS node detects the changes automatically and attempts to start.
- If the problem is resolved by updating the node properties, you must redeploy the flow before it can connect successfully.

Description of problem: A JMS node is unable to connect for a JMS provider and issues message BIP4648.

Corrective action:

1. Verify that the JMS provider server is running. If it is offline, start the server.

2. Verify that the JMS provider server is available from the integration node environment.
3. Ensure that the JMS provider Java `.jar` files have been placed into the integration node shared-classes directory on distributed systems, or that on z/OS, that these `.jar` files have been defined to the integration node CLASSPATH and any native libraries defined in the integration node LIBPATH.

The JMS nodes continue to attempt to connect to the JMS provider. Correct any problems and the JMS node should automatically detect the changes and attempt to connect to the provider.

Description of problem: A JMS node is unable to obtain a JMS destination and issues message BIP4642.

Corrective action

1. Investigate the cause of the problem described by the JMS exception message that might be included in the BIP event message.
2. Check that the name of the JMS destination that is defined in the relevant node property (Topic, Source Queue or Destination Queue) has been defined correctly in the JNDI administered objects.
3. Verify that the underlying system resource that is used by the JMS provider for the JMS destination has been configured correctly

Description of problem: A JMS input node does not attempt to reconnect to a JMS provider following a connection failure, or a restart of the JMS provider.

Corrective action: If the JMS provider is implemented by using a model that pushes on the JMS client, rather than a traditional polling model, the JMS provider might not throw an exception when calling receive() on an integration node connection. To resolve this problem, set the **`Handle connection exceptions asynchronously`** property of the JMS Providers policy to `True` for this JMS provider.

Processing MQTT messages

MQ Telemetry Transport (MQTT) is a lightweight publish/subscribe messaging protocol. IBM App Connect Enterprise provides built-in input and output nodes for processing MQTT messages.

The MQTT messaging protocol provides robust messaging features for communicating with remote systems and devices, and also minimizes network bandwidth and device resource requirements. The protocol is designed for devices in constrained environments, such as embedded systems, cell phones, and sensors with limited processing ability and memory, and for systems that are connected to unreliable networks. MQTT uses the publish/subscribe style of messaging that enables the information provider (publisher) to be decoupled from the consumer of the information (subscriber). Consequently, the MQTT protocol is ideal for the machine-to-machine, or Internet of Things, world of communication.

The MQTT V3.1 Protocol Specification is defined in the following document:

- MQTT Protocol Specification

For information on how to use MQTT with IBM App Connect Enterprise, see the following topics:

- “Using MQTT with IBM App Connect Enterprise” on page 887
- “Quality of service and connection management” on page 888
- “Using local environment variables with MQTT nodes” on page 890

Using MQTT with IBM App Connect Enterprise

Use IBM App Connect Enterprise to connect to applications and devices that send and receive messages by using the MQ Telemetry Transport (MQTT) messaging protocol.

You can use the built-in MQTTSubscribe and MQTTPublish nodes in IBM App Connect Enterprise to support the following operations:

- Receive a message that is published to one or more topics hosted on an MQTT server.
- Publish a message, from within a flow, to a topic hosted on an MQTT server.

All MQTT messages must conform to the MQTT Protocol Specification.

You can create message flows to receive an MQTT message by using the MQTTSubscribe node to subscribe to one or more topics on an MQTT server. You can send an MQTT message by using the MQTTPublish node in your message flow to publish messages to a topic on an MQTT server. In receiving and sending messages, the MQTT nodes behave like MQTT clients.

For more information on how to use the MQTT nodes in a message flow, including how to use the **Security identity** node property to connect to an MQTT server that requires authentication, see the following reference topics:

- MQTTSubscribe node
- MQTTPublish node

An example of how an MQTT application or device can be integrated with other technologies by using IBM App Connect Enterprise is available on GitHub as part of the mqtt-client-connector project. The example includes a simple Java application that simulates a blood pressure monitor, and a message flow that performs content-based routing. The message flow includes an MQTTSubscribe node to subscribe to the messages published by the blood pressure monitor application, and MQTTPublish nodes to publish blood pressure monitor readings to different topics. For more details on this example, see the MQTT Connector Sample on GitHub.

Client IDs for MQTT connections

The client ID is a string, 1 - 23 characters long, which identifies an MQTT client to an MQTT server. Each client that is connecting to a single server must use a unique client ID. If a client attempts to connect to an MQTT server with a client ID that is already in use, the server disconnects the first client connection, and the new connection is accepted instead.

When you develop a message flow, you must assign a unique client ID to each MQTT node in that message flow. Client IDs can be managed operationally by overriding the value in the BAR file, or by attaching a node policy. For more information, see command and “Overriding properties at run time with policies” on page 324. At deployment time, you must ensure that any MQTT nodes that are connecting to the same server have unique client IDs.

If you specify a client ID that is longer than 23 characters, only the last 23 characters are used when the MQTT node connects to the MQTT server.

Scalability

To increase the rate at which a message flow can process messages that are received by an MQTTSubscribe node, use the **Additional instances** property. This property increases the number of threads that the node can use to process messages, within a single integration server. Increasing the value of this property does not affect the number of input connectors available to receive messages.

Deploying the message flow across extra integration servers does not improve scalability, for the following reasons:

- The MQTTSubscribe node in each integration server will use the same client ID, and the MQTT server only allows one instance to connect with a particular client ID.
- If you configure each node instance to use a unique client ID, and to subscribe to the same topic, every message that is published to that topic is sent to all subscribers.

Quality of service and connection management
In MQTT, quality of service (QoS) controls the assurance of message delivery over the underlying transport. IBM App Connect Enterprise handles messages in different ways, depending on whether the QoS level is 0, 1, or 2.

The three levels of QoS are defined in the MQTT Protocol Specification. The following list provides a summary of the three levels:

QoS 0
At most once delivery (fire and forget)

QoS 1
At least once delivery (acknowledged delivery)

QoS 2
Exactly once delivery (assured delivery)

The open source Eclipse Paho MQTT client, which implements the MQTT protocol specification, is embedded in IBM App Connect Enterprise. The MQTT nodes use the Paho client to handle the exchange of MQTT protocol messages that are associated with the different QoS levels.

When IBM App Connect Enterprise connects to an MQTT server to publish a message with QoS 1 or QoS 2, the clean session flag is set to 0 (false). Therefore, any publications that are in flight when a connection is lost are delivered when IBM App Connect Enterprise reconnects to the server. For subscriptions, the clean session flag is set to 1 (true). When an MQTTSubscribe node connects, or reconnects, to an MQTT server, the connection is treated as a new connection.

Behavior of the MQTTSubscribe node

The QoS level for received messages applies only to the transfer of the messages from the MQTT server to IBM App Connect Enterprise. When a message with QoS 0 is successfully received by the Paho client, the MQTTSubscribe node propagates the message to the Out terminal. When a message with QoS 1 or QoS 2 is successfully received, the Paho client must acknowledge the receipt of the message before it can be processed by the node. After the transfer is successfully acknowledged and completed, the QoS level has no further effect on how the message is handled by IBM App Connect Enterprise.

If the network connection to the MQTT server is lost, the MQTTSubscribe node retries the connection after 5 seconds. The interval between reconnection attempts increases by 5 seconds on each attempt, up to a maximum of 2 minutes. There is no limit to the number of reconnection attempts. These attempts are logged in the MQTT Activity Log.

If messages arrive faster than they can be processed by IBM App Connect Enterprise, message receipt is blocked until the message flow is able to process them.

When an MQTTSubscribe node receives a message, the local environment is populated with the QoS of the message by using the **QualityOfService** variable. For more information about the local environment variables, see MQTTSubscribe node.

Behavior of the MQTTPublish node

The QoS level for published messages applies to the transfer of the message from IBM App Connect Enterprise to the MQTT server. When the MQTTPublish node publishes a message with QoS 0, an acknowledgment of the transfer is not expected. When the MQTTPublish node publishes a message with QoS 1 or QoS 2, the node waits for up to 30 seconds for the Paho client to confirm that the MQTT server has acknowledged receipt of the message. If no acknowledgment is received after this time, the node retries the publish. If this second attempt fails, an exception is produced with message BIP12099, and the node propagates the message to the Failure terminal if the node has a Failure terminal connected. If a Failure terminal is not connected, the message is rolled back to the input node.

Diagnosing transport issues

A connection failure at the network level generates an exception with message BIP12096. The MQTT resource statistics measurement **FailedConnections**, which reports the total number of attempted connections to an MQTT server that have failed since the last integration server restart, is also incremented.

The following connection issues are logged in the MQTT Activity Log:

- Failed to connect to an MQTT server
- Reattempted the connection to an MQTT server
- Lost the connection to an MQTT server
- Failed to disconnect from an MQTT server

The Activity Log also records both successful and unsuccessful attempts to publish a message to a topic, together with the QoS of the message. When an MQTTSubscribe node successfully subscribes to a topic, or receives a message from a topic, this success is also logged, together with the QoS of the subscription or the message.

Securing MQTT connections
The connection between MQTT nodes in a message flow and the MQTT server can be encrypted by using SSL.

Before you can encrypt the connection to the MQTT server, you must complete the following tasks:

- Create a truststore (if a truststore does not already exist), and import the MQTT server public certificate into the truststore.
- Configure the integration server to use the truststore.
- Obtain a user name and password (from the MQTT server administrator) that you can use to connect to the MQTT server.
- Set a value for the Security identity property on the MQTT message flow node.
- Link the Security identity property on the MQTT message flow node with the user name and password by using the **mqsisetdbparms** command; see command.

You can encrypt the connection between the MQTT server and a MQTTSubscribe or MQTTPublish message flow node by using one the following methods:

During development
Select the **Use SSL** check box in the MQTT message flow node properties, and set the appropriate port number. The default SSL port number is 8883.

During deployment
Set the connectionUrl property by using the **mqsiapplybaroverride**. The connectionUrl property must include, the protocol, the host name, and the port. For example:

```
mqsiapplybaroverride -b my.bar -k myApplication -m myFlow#MQTTPublish.connectionUrl=ssl://
myMQTTserver.com:8883
```

To configure the connection so that SSL is not used, change the protocol to tcp. For example:

```
mqsiapplybaroverride -b my.bar -k myApplication -m myFlow#MQTTPublish.connectionUrl=tcp://
myMQTTserver.com:1883
```

If set, the value of the connectionUrl property takes precedence over the values of the following MQTT message flow node properties:

- Host name
- Port
- Use SSL

For more information about the **mqsiapplybaroverride**, see command.

At run time
Set the connectionUrl local environment variable in the LocalEnvironment.Destination.MQTT.Output subtree; see “Using local environment variables with MQTT nodes” on page 890.

Using local environment variables with MQTT nodes
You can find the values that were used by the MQTTSubscribe and MQTTPublish nodes to process the MQTT message. You can also use fields in the local environment to dynamically alter the behavior of the MQTTPublish node.

These fields are available in the following message tree structures:

- LocalEnvironment
- LocalEnvironment.Destination

- LocalEnvironment.WrittenDestination

LocalEnvironment fields

When you use the MQTTSubscribe node, it stores information that you can access in the LocalEnvironment.MQTT.Input message tree. The fields in this structure are described in the following table.

Table 16. List of elements in the LocalEnvironment.MQTT.Input subtree

Element Name	Element Data Type	Description
Duplicate	BOOLEAN	Whether the message is a duplicate of a previous message. Set to TRUE or FALSE.
Retained	BOOLEAN	Whether the message is a retained message. Set to TRUE or FALSE. Retained is set to TRUE if the message was kept by the server and is now being sent when the client first connects to the server.
Topic	CHARACTER	The name of the MQTT topic the received message was published to.
QualityOfService	INTEGER	Quality of service of the received message. Set to 0 (at most once), 1 (at least once), or 2 (exactly once).

This structure is populated with each message written to the Out terminal of the MQTTSubscribe node.

LocalEnvironment.Destination fields

When you use the MQTTPublish node, you can override certain properties with elements in the LocalEnvironment.Destination.MQTT.Output message tree. The fields in this structure are described in the following table.

Table 17. List of elements in the LocalEnvironment.Destination.MQTT.Output subtree

Element Name	Element Data Type	Description
retained	BOOLEAN	Whether the message is a retained message. Set to TRUE or FALSE. The default is FALSE. Set to TRUE if the message for a topic must be held by the MQTT server after delivery to all currently connected clients, and then delivered to new clients when they connect to that topic. Each retained message for a topic replaces the previous retained message for that topic. In this way, devices can receive the most recent retained message immediately on connecting to a topic.
TopicName	CHARACTER	The name of the MQTT topic the message will be published to.
qos	INTEGER	Quality of service of the published message. Set to 0 (at most once), 1 (at least once), or 2 (exactly once).

Table 17. List of elements in the LocalEnvironment.Destination.MQTT.Output subtree (continued)		
Element Name	**Element Data Type**	**Description**

connectionUrl	CHARACTER	The URL that is used to connect to the MQTT server. The URL includes the protocol, the host name, and the port. For example:
		`tcp://myMQTTserver.com:1883`
		or, if the connection is encrypted:
		`ssl://myMQTTserver.com:8883`
		If set, the value of the connectionUrl property takes precedence over the values of the following MQTT message flow node properties:
		• Host name
		• Port
		• Use SSL

LocalEnvironment.WrittenDestination fields

When you use the MQTTPublish node, it stores information that you can access in the LocalEnvironment.WrittenDestination.MQTT message tree. The fields in this structure are described in the following table.

Table 18. List of elements in the LocalEnvironment.WrittenDestination.MQTT subtree		
Element Name	**Element Data Type**	**Description**
ClientId	CHARACTER	The unique name of the client.
DeliveryToken.isComplete	BOOLEAN	Whether the message was successfully published to the MQTT topic. Set to TRUE or FALSE.

This structure is populated with each message written to the Out terminal of the MQTTPublish node.

Processing Kafka messages

Apache Kafka is an open source project that provides a messaging service capability, based upon a distributed commit log, which lets you publish and subscribe data to streams of data records (messages). IBM App Connect Enterprise provides built-in input and output nodes for processing Kafka messages. A mid-flow node is also provided for reading individual messages on a Kafka topic.

The Kafka messaging protocol is a TCP based protocol that provides a fast, scalable, and durable method for exchanging data between applications. Messaging providers are typically used in IT architectures in order to decouple the processing of messages from the applications that produce them. IBM App Connect Enterprise has very strong existing support for many kinds of messaging, including IBM MQ, JMS messaging providers, and the MQTT messaging protocol. The Kafka nodes expand this support to help IBM App Connect Enterprise interact with Kafka applications.

For information about the supported versions of Kafka, see IBM App Connect Enterprise system requirements. For more information about Kafka version compatibility, see the Apache Kafka documentation.

Kafka is a very popular choice for cloud-based architectures, where the numbers of connected clients can change frequently without impacting the scaling characteristics. Common use-cases where Kafka is considered as a messaging transport include:

General messaging
You can decouple producer and consumer applications from each other. Data flowing between multiple processing stages in a distributed application could use Kafka as a means of connecting the steps. The built-in features that Kafka provides for partitioning, replication, and fault tolerance can make it a good choice for this type of large-scale messaging.

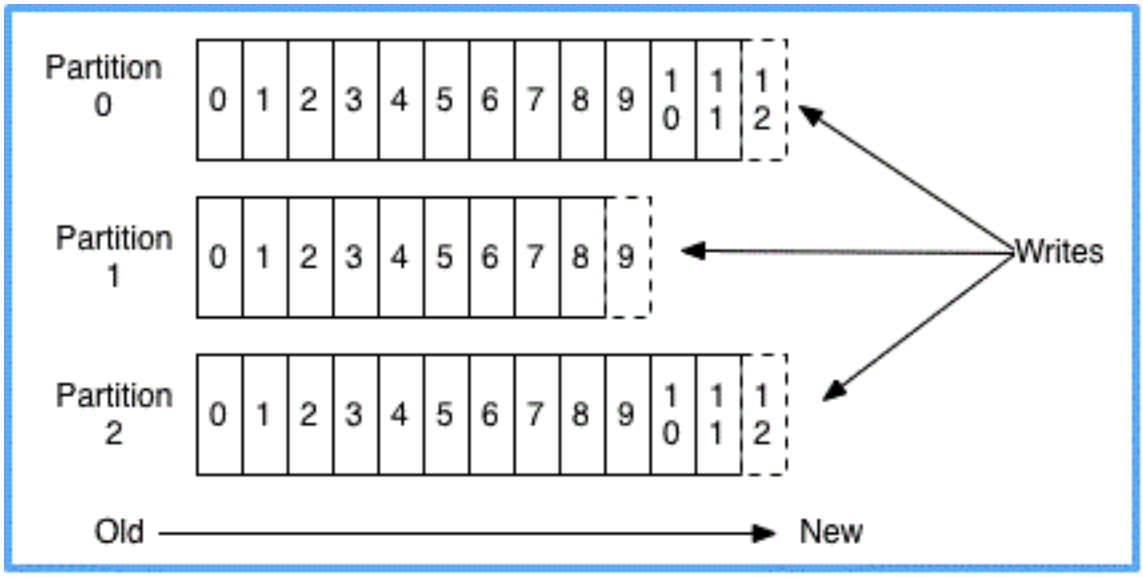
Web site activity tracking
When Kafka was first developed, it was used for helping to track page views, searches, or other actions taken on a web site. This activity was published to a set of central topics for different activity types. Subscribers could then use the data for real-time processing and monitoring, and persisting to other data warehouse systems.

Logging and metrics
You can use Kafka to aggregate operational data from multiple sources.

Apache Kafka maintains feeds of messages in categories called topics. IBM App Connect Enterprise provides built-in nodes for processing Kafka messages, which use the Apache Kafka Java client:

- node, which publishes messages to a Kafka topic
- node, which subscribes to a Kafka topic and propagates the feed of published messages to nodes connected downstream in the flow
- node, which reads a specified message from a Kafka topic.

IBM App Connect Enterprise acts as a Kafka client, and can communicate with your Kafka implementation by sending messages over the network to the Kafka cluster. A Kafka cluster implementation is made up of one or more servers, known as Kafka brokers. Kafka brokers replicate data between themselves in order to cope with specific server failures without losing any messages that have been committed to the Kafka log. Each instance of a Kafka node is configured with the name of a topic. For each topic maintained by Kafka, the Kafka brokers maintain a partitioned commit log, as shown in the following diagram:



Each partition is an ordered sequence of messages, whose state cannot be altered after they have been created. Each of the messages in a partition is assigned a sequential ID number, called the *offset*, which uniquely identifies each message in the partition. These messages are all retained for a configurable period of time, regardless of whether they have been consumed by another application, after which the messages are discarded to free up space. This approach to message queuing is very different from the approach of traditional messaging products such as IBM MQ.

Message ordering is preserved only within a partition, not across all topics in a partition; therefore, if message order is important, ensure that you either use a single partition per topic, or associate a message

key with each message published. A hash of the key for a message is used to select the partition to which the message is sent, so all messages published with the same key are stored on the same partition.

For more information about Kafka messaging, see the Apache Kafka documentation.

For information about how to use Kafka with IBM App Connect Enterprise, see the following topics:

- "Using Kafka with IBM App Connect Enterprise" on page 894
- "Producing messages on Kafka topics" on page 895
- "Consuming messages from Kafka topics" on page 898
- "Reading an individual message from a Kafka topic" on page 901
- "Configuring security credentials for connecting to Kafka" on page 904
- "Using local environment variables with Kafka nodes" on page 907
- "Setting and retrieving Kafka custom header properties" on page 903
- "Using Kafka nodes with IBM Event Streams" on page 909
- "Authenticating connections to a Kafka cluster by using SASL/SCRAM" on page 905
- "Resolving problems when using Kafka nodes " on page 2941

Using Kafka with IBM App Connect Enterprise

You can use the IBM App Connect Enterprise Kafka nodes to produce and consume messages on Kafka topics.

Before you begin

- Read the topic "Processing Kafka messages" on page 892

About this task

You can use the node to publish messages that are generated from within your message flow to a topic that is hosted on a Kafka server. The published messages are then delivered by the Kafka server to all topic consumers (subscribers). You can use a node in a message flow to subscribe to a specified topic on a Kafka server. The KafkaConsumer node then receives messages that are published on the Kafka topic, as input to the message flow. You can also read an individual message that is published on a Kafka topic, by using a node in a message flow to specify the offset position of the message in the topic partition.

For more information about the supported versions of Kafka, see IBM App Connect Enterprise system requirements. For more information about Kafka version compatibility, see the Apache Kafka documentation.

Procedure

Use the KafkaProducer, KafkaConsumer, and KafkaRead nodes to complete the following tasks:

- "Producing messages on Kafka topics" on page 895.
- "Consuming messages from Kafka topics" on page 898.
- "Reading an individual message from a Kafka topic" on page 901.

For more information about configuring and using Kafka nodes, see the following topics:

- "Using local environment variables with Kafka nodes" on page 907.
- "Setting and retrieving Kafka custom header properties" on page 903,
- "Using Kafka nodes with IBM Event Streams" on page 909.
- "Configuring security credentials for connecting to Kafka" on page 904.
- "Authenticating connections to a Kafka cluster by using SASL/SCRAM" on page 905.
- Kafka policy.
- "Resolving problems when using Kafka nodes " on page 2941.

Producing messages on Kafka topics
You can use the KafkaProducer node to connect to the Kafka messaging system and publish messages on Kafka topics.

Before you begin

Read the following topics:

- "Processing Kafka messages" on page 892
- "Using Kafka with IBM App Connect Enterprise" on page 894

About this task

You can use the KafkaProducer node to publish messages that are generated from within your message flow to a topic that is hosted on a Kafka server. The published messages are then available to be received by consumers (subscribers) reading from the topic.

You can use a KafkaConsumer node in a message flow to subscribe to a specified topic on a Kafka server. You can also use a KafkaRead node to read an individual message on a specified topic. For more information about using these nodes, see "Consuming messages from Kafka topics" on page 898 and "Reading an individual message from a Kafka topic" on page 901.

The KafkaProducer node publishes messages non-transactionally to the Kafka server, and they are available to be read by consuming applications as soon as they are published. Because the publish operation is non-transactional, if the flow is rolled back after the message passes through the KafkaProducer node, the publication of the message to the Kafka server is not rolled back. However, you can use the **Acks** property on the KafkaProducer node to configure synchronous processing of the message, by specifying that the KafkaProducer node must wait for confirmation that the message was successfully received by the Kafka server before continuing in the flow. The options that you can specify for the **Acks** property are described in the steps of this task topic.

You can use Kafka custom header properties to add metadata to Kafka messages for use during message processing. These properties are set in the LocalEnvironment, in a folder called `KafkaHeader`. For more information, see "Setting and retrieving Kafka custom header properties" on page 903.

Procedure

Complete the following steps to use IBM App Connect Enterprise to publish messages to a topic on a Kafka server:

1. Create a message flow that contains an input node, such as an HTTPInput node, and a KafkaProducer node.

   For more information about how to create a message flow, see "Creating a message flow" on page 574.
2. Configure the KafkaProducer node by setting the following properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Topic name` property, specify the name of the Kafka topic on which you want to publish messages.

        The topic name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash). The topic name can be changed dynamically

by setting a local environment override. For more information, see “Using local environment variables with Kafka nodes” on page 907.

- In the `Bootstrap servers` property, specify the hostname and port of the Kafka server that you want to connect to; for example, if you are using IBM Event Streams (Kafka on IBM Cloud), specify the address of that server.
- In the `Client ID` property, specify the client name to be used when connecting to the Kafka server.

  The client name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
- Use the `Add server name suffix to client ID` property to specify whether you want to suffix the client ID with the name of the integration server and integration node. This property is

selected by default, and adds the integration server and integration node name to the end of the client ID, in the following format:

```
'integration_server_name'-'integration_node_name'
```

- In the Acks property, select the number of acknowledgments that are expected from the kafka cluster in order to consider the message successfully published.

  If this property is set to 0, the KafkaProducer node does not wait for any acknowledgment that the publish request has been processed by the Kafka server. This is equivalent to a 'fire and forget' mode of operation.

  If this property is set to 1, the KafkaProducer node waits for a single acknowledgment from the Kafka server.

  If this property is set to All, the KafkaProducer node waits for acknowledgments from all replicas of the topic. This option provides the strongest available guarantee that the record was received.
- In the Timeout property, specify the time (in seconds) to wait for a request to complete. The default value is 60 seconds.

b) On the **Security** tab, set the following properties:

- In the Security protocol property, select the protocol to be used when communicating with the integration node. Valid values are:
  - PLAINTEXT
  - SSL
  - SASL_PLAINTEXT
  - SASL_SSL

  The default value for this property is PLAINTEXT.

  Note: If you are using Event Streams, this property must be set to SASL_SSL.

  If either SASL_PLAINTEXT or SASL_SSL is selected, you must configure the user ID and password to use for authenticating with the Kafka server by configuring the security identity. For more information, see "Configuring security credentials for connecting to Kafka" on page 904.
- In the SSL protocol property, select the SSL protocol to be used if the Security protocol property is set to either SSL or SASL_SSL. You can select one of the following values from the editable list, or you can specify an alternative value:
  - TLSv1
  - TLSv1.1
  - TLSv1.2
  - TLSv1.3

  The default value for this property is TLSv1.2.

For more information about other properties that you can set for the KafkaProducer node, see node. For more information about setting properties by using a policy, see Kafka policy.

For more information about how to diagnose connection problems between IBM App Connect Enterprise and Kafka, see "Resolving problems when using Kafka nodes " on page 2941.

What to do next

For more information about properties that can be overridden dynamically in the flow, see "Using local environment variables with Kafka nodes" on page 907.

Consuming messages from Kafka topics
You can use the KafkaConsumer node to receive messages that are published on a Kafka topic.

Before you begin

Read the following topics:

- "Processing Kafka messages" on page 892
- "Using Kafka with IBM App Connect Enterprise" on page 894

About this task

You can use a KafkaConsumer node in a message flow to subscribe to a specified topic on a Kafka server. The KafkaConsumer node then receives messages that are published on the Kafka topic, as input to the message flow.

You can use a KafkaProducer node to publish messages from your message flow to a topic that is hosted on a Kafka server, and you can use a KafkaRead node to read an individual message on a Kafka topic. For more information about using these nodes, see "Producing messages on Kafka topics" on page 895 and "Reading an individual message from a Kafka topic" on page 901.

Each KafkaConsumer node consumes messages from a single topic; however, if the topic is defined to have multiple partitions, the KafkaConsumer node can receive messages from any of the partitions. For more information about partitions in Kafka topics, see the Apache Kafka documentation.

The KafkaConsumer node reads messages from Kafka non-transactionally, which means that, if an error occurs or the message is rolled back to the input node, and no catch terminal is connected, the message is not reprocessed by the input node.

In order to process messages that are received concurrently, you can configure additional instances on the KafkaConsumer node. When additional instances are configured, a single Kafka message consumer is created, and the messages are distributed to the additional flow instances. As messages are processed concurrently, message ordering is not preserved when additional instances are being used. For more information about specifying additional instances, see node.

You can also increase concurrency by deploying multiple KafkaConsumer nodes that share the same **`Group ID`**; Kafka ensures that messages that are published on the topic are shared across the consumer group. For more information about how Kafka shares the message across multiple consumers in a consumer group, see the Apache Kafka documentation.

You can use Kafka custom header properties to add metadata to Kafka messages for use during message processing. These properties are set in the LocalEnvironment, in a folder called `KafkaHeader`. For more information, see "Setting and retrieving Kafka custom header properties" on page 903.

Procedure

Complete the following steps to receive messages that are published on a Kafka topic:

1. Create a message flow containing a KafkaConsumer node and an output node.

2. Configure the KafkaConsumer node by setting the following properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Topic name` property, specify the name of the Kafka topic to which you want to subscribe.

        The topic name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
      - In the `Bootstrap servers` property, specify the host name and port of the Kafka server; for example, if you are using IBM Event Streams (Kafka on IBM Cloud), specify the address of that server.
      - In the `Consumer group ID` property, specify the ID of the consumer group to which this consumer belongs. This ID can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
      - In the `Default message offset` property, specify the message offset that is used by the consumer when it starts, if no valid message offset exists; for example, when the consumer starts for the first time, or when the consumer is not committing its offset to Kafka. Possible values are:
        - `Earliest`
        - `Latest`

        This property defines the starting position in the log of messages from where the KafkaConsumer node starts reading messages. The default setting is `Latest`, which means that only messages published after the flow is started will be read.

        If `Earliest` is selected, the KafkaConsumer node starts reading messages from the first message in the log of messages for the specified topic.

        If the `Commit message offset in Kafka` property is not selected, this action is repeated each time the flow containing the KafkaConsumer node is started. If the `Commit message offset in Kafka` property is selected, the consumer position in the log of messages for the topic is saved in Kafka as each message is processed; therefore, if the flow is stopped and then restarted, the input node starts consuming messages from the message position that had been reached when the flow was stopped.
      - Use the `Commit message offset in Kafka` property to specify whether the current message offset in Kafka is saved automatically, which allows messages to be consumed from the saved position when the consumer is restarted. This property is selected by default. If the integration server or integration node is restarted, the last saved position is used.
      - Use the `Wait for message offset commit to complete` property to control whether the KafkaConsumer node waits for the message offset to be committed to Kafka before delivering the message to the message flow. If this property is not selected, the KafkaConsumer node does not wait for the response from the commit operation to be received before delivering the message to the flow. As a result, message throughput is increased in exchange for a reduction in message reliability, as messages can be redelivered to the message flow if the request to commit the consumer offset subsequently fails. This property is selected by default.
      - In the `Client ID` property, specify the client name to be used when connecting to the Kafka server.

        The client name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
      - Use the `Add server name suffix to client ID` property to specify whether you want to suffix the client ID with the name of the integration server and integration node. This property is

selected by default, and adds the integration server and integration node name to the end of the client ID, in the following format:

```
'integration_server_name'-'integration_node_name'
```

b) On the **Advanced** tab, set the following properties:

- In the `Connection timeout (sec)` property, specify the maximum time that the KafkaConsumer node will wait to establish a connection with the Kafka server. The default value for this property is 15 seconds.

  Note: The value specified for the `Connection timeout` must be greater than the value specified for the `Session timeout`.
- In the `Session timeout (sec)` property, specify the maximum time that the Kafka server should wait to receive confirmation (in the form of periodic 'heartbeats') that the KafkaConsumer node is live. This property is used to detect KafkaConsumer node failures when using Kafka's group management facility. The KafkaConsumer node sends periodic heartbeats to indicate its liveness to the Kafka server. If no heartbeats are received by the Kafka server before the expiration of this session timeout, the Kafka server removes this consumer from the group and initiates a rebalance. The minimum valid value for this property is `10` seconds, which ensures that the session timeout is greater than the length of time between heartbeats. The `Session timeout` value must be less than the `Connection timeout` value.
- In the `Receive batch size` property, specify the maximum number of records that are received from the Kafka server in a single batch. This property is used for the *max.poll.records* value specified by the KafkaConsumer node when receiving messages from the Kafka server.

c) On the **Security** tab, set the following properties:

- In the `Security protocol` property, select the protocol to be used when communicating with the integration node. Valid values are:
  - `PLAINTEXT`
  - `SSL`
  - `SASL_PLAINTEXT`
  - `SASL_SSL`

  The default value for this property is `PLAINTEXT`.

  Note: If you are using Event Streams, this property must be set to `SASL_SSL`.

  If either `SASL_PLAINTEXT` or `SASL_SSL` is selected, you must configure the user ID and password that will be used to authenticate with the Kafka server by configuring the security identity, as described in “Configuring security credentials for connecting to Kafka” on page 904.
- In the `SSL protocol` property, select the SSL protocol to be used if the `Security protocol` property is set to either `SSL` or `SASL_SSL`. You can select one of the following values from the editable list, or you can specify an alternative value:
  - TLSv1
  - TLSv1.1
  - TLSv1.2
  - TLSv1.3

  The default value for this property is `TLSv1.2`.

For information about other properties that you can set for the KafkaConsumer node, see node. For information about setting properties by using a policy, see Kafka policy.

For more information about how to diagnose connection problems between IBM App Connect Enterprise and Kafka, see “Resolving problems when using Kafka nodes ” on page 2941.

What to do next

For information about properties that can be overridden dynamically in the flow, see "Using local environment variables with Kafka nodes" on page 907.

Reading an individual message from a Kafka topic

You can use the KafkaRead node to read an individual message published on a Kafka topic, by specifying the offset position of the message in the topic.

Before you begin

Read the following topics:

- "Processing Kafka messages" on page 892
- "Using Kafka with IBM App Connect Enterprise" on page 894

About this task

You can use a node to help with error processing in a Kafka flow. For example, if the processing of a message fails in a Kafka consumer message flow, you can use the KafkaRead node to reprocess that individual message. You can also use a KafkaRead node in a flow with a Timer node to control the processing of a range of messages in a partition on a topic. For more information, see Processing Kafka messages.

You can specify a message in a partition on a Kafka topic, by using the **partition** property on the KafkaRead node. For more information about partitions in Kafka topics, see the Apache Kafka documentation.

For more information about subscribing to topics on a Kafka server by using a node, see "Consuming messages from Kafka topics" on page 898. For more information about publishing messages to a Kafka topic by using a node, see "Producing messages on Kafka topics" on page 895.

By default, if the message at the specified offset is not available, the input message is sent to the No match terminal. Alternatively, you can choose to read either the earliest or the latest message in the partition if the specified message is unavailable, by setting one of those values in the **Action when message unavailable** property of the KafkaRead node. You can also override this value dynamically by using the command.

Procedure

Complete the following steps to read an individual message that is published on a Kafka topic:

1. Create a message flow that contains an input node, a KafkaRead node, and an output node.
   For more information about how to do this, see "Creating a message flow" on page 574.
2. Configure the KafkaRead node by setting the following properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Topic name` property, specify the name of the Kafka topic that contains the message that you want to read.

        The topic name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
      - In the `Partition number` property, specify the number of the Kafka partition for the topic that you want to use (valid values are between `0` and 255). The default value is `0`.
      - In the `Bootstrap servers` property, specify the host name and port to be used for establishing the initial connection to the Kafka cluster; for example, if you are using IBM Event Streams (Kafka

on IBM Cloud), specify the address of that server. You can specify multiple servers by defining a list of host/port pairs, separated by commas.

- In the `Message offset` property, specify the offset position of the message to be read. This property defines the position in the log of messages from where the KafkaRead node will read the message. The default value is `0`.
- In the `Action when message unavailable` property, select the action that will be taken if the message at the specified offset is not available:
  - Select `no match` to propagate the input message to the `No match` terminal. This is the default value.
  - Select `earliest` to read the message with the earliest offset for the topic partition.
  - Select `latest` to read the message with the latest offset for the topic partition.
- In the `Client ID` property, specify the client name to be used when connecting to the Kafka server.

  The client name can be up to 255 characters in length, and can include the following characters: a-z, A-Z, 0-9, . (dot), _ (underscore), and - (dash).
- Use the `Add server name suffix to client ID` property to specify whether you want to suffix the client ID with the name of the integration server. This property is selected by default, and adds the integration server name to the end of the client ID, in the following format:

  ```
  -integrationServerName
  ```

  for an independent integration server, or

  ```
  integration_node_name-integration_server_name
  ```

  for an integration server that is managed by an integration node.
- In the `Timeout (s)` property, specify the length of time (in seconds) in which the KafkaRead node must read the message. If the message has not been read within this timeout period, the action specified by the `Action when message unavailable` property is taken.

b) On the **Security** tab, set the following properties:

- In the `Security protocol` property, select the protocol to be used when communicating with the Kafka server. Valid values are:
  - `PLAINTEXT`
  - `SSL`
  - `SASL_PLAINTEXT`
  - `SASL_SSL`

  The default value for this property is `PLAINTEXT`.

  If you are using Event Streams, this property must be set to `SASL_SSL`.

  If either `SASL_PLAINTEXT` or `SASL_SSL` is selected, you must configure the user ID and password that will be used to authenticate with the Kafka server by configuring the security identity, as described in "Configuring security credentials for connecting to Kafka" on page 904.
- In the `SSL protocol` property, select the SSL protocol to be used if the `Security protocol` property is set to either `SSL` or `SASL_SSL`. You can select one of the following values from the editable list, or you can specify an alternative value:
  - TLSv1
  - TLSv1.1
  - TLSv1.2
  - TLSv1.3

  The default value for this property is `TLSv1.2`.

For information about other properties that you can set for the KafkaRead node, see node. For information about setting properties by using a policy, see Kafka policy.

For more information about how to diagnose connection problems between IBM App Connect Enterprise and Kafka, see “Resolving problems when using Kafka nodes ” on page 2941.

What to do next

For information about properties that can be overridden dynamically in the flow, see “Using local environment variables with Kafka nodes” on page 907.

Setting and retrieving Kafka custom header properties

You can use Kafka custom header properties to associate metadata with a Kafka message.

Before you begin

Read the following topics:

- “Processing Kafka messages” on page 892
- “Using Kafka with IBM App Connect Enterprise” on page 894

About this task

Kafka custom header properties enable you to add metadata to the Kafka message, which can then be used during message processing; for example, the header properties can carry information about the format of the data, such as a schema name.

Kakfa custom header properties are a set of `name:value` pairs, which can be associated with a Kafka message as it is published and then retrieved when the message is retrieved. IBM App Connect Enterprise supports Kafka header properties that consist of a name and a value, both of which must be strings in Java String format.

The properties are set in the Local Environment, in a folder called `KafkaHeader`. For the KafkaProducer node, all property values must be supplied as a string in the Local Environment. For the KafkaConsumer and KafkaRead nodes, the values of header properties in the received message are converted to string values when they are written into the Local Environment.

Before calling the KafkaProducer node, you can set the Kafka custom header properties in the `Environment.Destination.Kafka.Output.KafkaHeader` Local Environment folder. For example:

```
# To set Kafka custom header properties called 'Name' and 'Occupation'
      SET OutputLocalEnvironment.Destination.Kafka.Output.KafkaHeader.Name = 'Bob';
      SET OutputLocalEnvironment.Destination.Kafka.Output.KafkaHeader.Occupation = 'Builder';
```

On return from a KafkaConsumer node, the Local Environment folder `LocalEnvironment.Kafka.Input` is updated with any custom headers properties contained in the received message, which can then be processed by using a Compute node. For example:

```
 # To retrieve the value of the custom header property called 'Name'
       SET Name = InputLocalEnvironment.Kafka.Input.KafkaHeader.Name;
```

On return from a KafkaRead node, the Local Environment folder `LocalEnvironment.Kafka.Read` is updated with any custom headers properties contained in the received message, which can then be processed by using a Compute node. For example:

```
# To retrieve the value of the custom header property called 'Occupation'
       SET Occupation = InputLocalEnvironment.Kafka.Read.KafkaHeader.Occupation;
```

For more information about the Kafka nodes, see the following topics:

- node
- node
- node

For information about properties that can be overridden dynamically in the flow, see "Using local environment variables with Kafka nodes" on page 907.

Configuring security credentials for connecting to Kafka

Use the **mqsisetdbparms** or **mqsicredentials** command to associate security credentials with a connection to a Kafka cluster, and configure the Kafka nodes to authenticate by using the required username and password.

Before you begin

Read the following topics:

- "Processing Kafka messages" on page 892
- "Using Kafka with IBM App Connect Enterprise" on page 894

About this task

Before you can connect to a Kafka cluster that requires authentication with a username and password, you must use either the **mqsisetdbparms** or **mqsicredentials** command to configure the credentials that the KafkaConsumer, KafkaRead, and KafkaProducer nodes use to authenticate to the Kafka cluster.

To configure the Kafka nodes to authenticate using the username and password, you set the **Security protocol** property on the node to either SASL_PLAINTEXT or SASL_SSL.

If you are using the IBM Event Streams service on IBM Cloud, the **Security protocol** property on the Kafka node must be set to SASL_SSL. For more information about configuring the security credentials for connecting to Event Streams, see "Using Kafka nodes with IBM Event Streams" on page 909.

For more information about configuring Kafka nodes to authenticate by using Salted Challenge Response Authentication Mechanism (SCRAM), see "Authenticating connections to a Kafka cluster by using SASL/SCRAM" on page 905.

Procedure

Follow these steps to configure a connection to a secured Kafka cluster:

1. Use either the **mqsisetdbparms** or **mqsicredentials** command to associate a username and password with a connection to a Kafka cluster:
   - Configure security credentials by using the **mqsisetdbparms** command, specifying the required username (**-u**), password (**-p**), and resource name (**-n**). The resource name is in the form `kafka::` followed by the name of the security identity that is specified on the Kafka node; for example, `kafka::myKafkaSecId`. Alternatively, you can use the default security identity, by specifying a resource name of `kafka::KAFKA` or `kafka::KAFKA::integrationServerName`.

     The following example shows how to specify a username, password, and named Kafka security identity:

     ```
     mqsisetdbparms -w workDir -n kafka::myKafkaSecId -u myUsername -p myPassword
     ```

     The following example shows how to specify a username and password, and specifies that the default Kafka security identity for the integration server will be used:

     ```
     mqsisetdbparms -w workDir -n kafka::KAFKA::myIntegrationServer1 -u myUsername -
     p myPassword
     ```

     For more information, see command.
   - Configure security credentials by using the **mqsicredentials** command, specifying the username (**--username**), password (**--password**), credential type (**--credential-type**), and credential name (**--credential-name**). Specify the credential type as `kafka`, and specify a credential name that matches the value of the **Security identifier** parameter that is specified in the node.

Alternatively, you can use the default security identity that is specified by the **--set-as-default** parameter on the **mqsicredentials** command.

The following example shows how to specify a username, password, and named Kafka security identity:

```
mqsicredentials --create --work-dir workDir --credential-type kafka --credential-name
 myKafkaSecId --username myUsername --password myPassword
```

The following example shows how to specify a username and password, and specifies that the default Kafka security identity for the integration server will be used:

```
mqsicredentials --create --work-dir workDir --credential-type kafka --credential-name
 myKafkaSecId --username myUsername --password myPassword
```

```
mqsicredentials --set-as-default --work-dir workDir --credential-type kafka --credential-
name myKafkaSecId
```

For more information, see command.

2. In your message flow, set the following properties on the KafkaConsumer, KafkaRead, and KafkaProducer nodes:
   a) Set the **Security protocol** property to either SASL_PLAINTEXT or SASL_SSL.

      If you are using the Event Streams service on IBM Cloud, the **Security protocol** property must be set to SASL_SSL.

   b) If either SSL or SASL_SSL is specified by the **Security protocol** property, you must also specify the **SSL protocol** to be used.

      You can select one of the following values from the supplied list, or you can specify an alternative value:

      - TLSv1
      - TLSv1.1
      - TLSv1.2
      - TLSv1.3

   c) In the **Security identifier** property, specify the name of the security credential that you created in step 1. Alternatively, if you configured a default security identity in step 1 (for example, myKafkaSecId), you can leave the **Security identifier** property empty and the default security identity is used.

   For more information about the Kafka nodes, see the following topics:

   - node
   - node
   - node

   For more information about how to diagnose connection problems between IBM App Connect Enterprise and Kafka, see “Resolving problems when using Kafka nodes ” on page 2941.

What to do next

You can use the command or the command to show information about the credentials that are being used for connecting to a Kafka cluster.

Authenticating connections to a Kafka cluster by using SASL/SCRAM

You can configure KafkaConsumer, KafkaRead, and KafkaProducer nodes to authenticate with a Kafka cluster by using Salted Challenge Response Authentication Mechanism (SCRAM) or SASL/SCRAM.

Before you begin

Read the following topics:

- “Processing Kafka messages” on page 892
- “Using Kafka with IBM App Connect Enterprise” on page 894

About this task

Salted Challenge Response Authentication Mechanism (SCRAM), also known as SASL/SCRAM, is an SASL mechanism that performs password-based authentication between the client and server, and resolves some of the security concerns that are associated with SASL_PLAIN authentication.

To authenticate a connection between Kafka nodes and a Kafka cluster that uses SCRAM for authentication, you must configure the nodes to use a Kafka policy that is configured with the connection details.

Procedure

Follow these steps to enable Kafka nodes to authenticate a connection to a Kafka cluster by using SCRAM:

1. Use either the command or the command to associate a username and password with a security identity.
   - Configure security credentials by using the **mqsisetdbparms** command, specifying the required username (**-u**), password (**-p**), and resource name (**-n**). The resource name is in the form `kafka::` followed by the name of the security identity that is specified on the Kafka node; for example, `kafka::myKafkaSecId`.

     The following example shows how to specify a username, password, and named Kafka security identity:

     ```
     mqsisetdbparms -w workDir -n kafka::myKafkaSecId -u myUsername -p myPassword
     ```

     For more information, see command.
   - Configure security credentials by using the **mqsicredentials** command, specifying the username (**--username**), password (**--password**), credential type (**--credential-type**), and credential name (**--credential-name**). Specify the credential type as `kafka`, and specify a credential name that matches the value of the **Security identifier** parameter that is specified in the node.

     The following example shows how to specify a username, password, and named Kafka security identity:

     ```
     mqsicredentials --create --work-dir workDir --credential-type kafka --credential-name
      myKafkaSecId --username myUsername --password myPassword
     ```

     For more information, see command.

2. Create a Kafka policy in a policy project by using the IBM App Connect Enterprise Toolkit, and set the following properties:
   a) Set the value of the **Bootstrap servers** property to the list of bootstrap servers for the Kafka cluster. You can specify a single *hostname:port* value or a comma-separated list of multiple *hostname:port* values.
   b) Set the **Security protocol** property to SASL_SSL.
   c) Set the **Security mechanism** property to either SCRAM-SHA-256 or SCRAM-SHA-512, as required by the Kafka cluster.
   d) Set the **Security identity** property to the name of the security identity that you created in step 1 (for example, myKafkaSecId).
   e) Set the **SASL config** property to `org.apache.kafka.common.security.scram.ScramLoginModule required;`

      This value specifies the SASL configuration to use when you connect to the Kafka cluster.
   f) Optional: Set the **SSL truststore location** to the location of the truststore that contains the public or CA certificate for the Kafka cluster.
   g) Optional: Set the **SSL truststore type** to the type of truststore that is specified in the **SSL truststore location** property. Possible values are JKS and PKCS12.
   h) Optional: Set the **SSL truststore security identity** to specify the security identity used to access the truststore.

   For more information, see Kafka policy.
3. On the **Policy** tab of the Kafka nodes, set the **Policy** property to the name of the policy created in step 1, in the form *{PolicyProjectName}:PolicyName*

   For more information about the Kafka nodes, see node, node, and node.

   For more information about how to diagnose connection problems between IBM App Connect Enterprise and Kafka, see "Resolving problems when using Kafka nodes " on page 2941.

Using local environment variables with Kafka nodes

The KafkaConsumer, KafkaRead, and KafkaProducer nodes support local environment message tree variables, which you can use to dynamically alter the node properties.

The following table shows the elements in the **LocalEnvironment.Destination.Kafka.Output** message tree, which can be used to override the **Topic name** property in the KafkaProducer node. The table includes an example of how to set the values by using ESQL; however, you can also set them by using transformation nodes, such as the Mapping node. To access the LocalEnvironment from a Mapping node, see Customizing a message map to include a message assembly component.

Table 19. Input local environment properties

Element name	Type	Description
topicName	string	The name of the topic where the message will be published. This environment variable overrides the **Topic name** property on the node. For example: `SET OutputLocalEnvironment.Destination.Kafka.Output.topicName = 'customers';`
key	string	A string value to associate with the message. When the message is being published to a topic with multiple partitions, a hash of the key value is used to select the partition on which the message is stored. If no key is provided, messages are distributed across the topic partitions by Kafka. All messages published using the same key value are sent to the same partition.

The following table shows the data that is written by the KafkaProducer node to the LocalEnvironment when propagating an output message:

Table 20. Output local environment properties generated by the KafkaProducer node		
Element	**Type**	**Description**
LocalEnvironment.WrittenDestination.Kafka.partition	string	The partition (in the topic) that contains the message.
LocalEnvironment.WrittenDestination.Kafka.topicName	string	The topic where the message was published.
LocalEnvironment.WrittenDestination.Kafka.offset	integer	The offset number in the partition, for the message that was published.
LocalEnvironment.WrittenDestination.Kafka.checksum	integer	The checksum of the message.
LocalEnvironment.WrittenDestination.Kafka.key	string	The key value that was provided in the input local environment.

After a KafkaConsumer node has read a message, these properties are put into the LocalEnvironment. The KafkaRead node can take these values as input for re-reading the message in a failure scenario for which the KafkaRead node is being used (during catch processing).

The following table shows the data that is written by the KafkaConsumer node to the LocalEnvironment when propagating an output message:

Table 21. Output local environment properties generated by the KafkaConsumer node		
Element	**Type**	**Description**
LocalEnvironment.Kafka.Input.partition	string	The partition (in the topic) that contains the message.
LocalEnvironment.Kafka.Input.topicName	string	The topic where the message was published.
LocalEnvironment.Kafka.Input.offset	integer	The offset number in the partition, for the message that was received.
LocalEnvironment.Kafka.Input.checksum	integer	The checksum of the message.
LocalEnvironment.Kafka.Input.key	string	The key (if any) that was associated with the received message. This field exists only if a key was associated with the message when it was published. The received key value is received as a string by the KafkaConsumer node.

The following table shows elements that can be used to override the properties in the KafkaRead node:

Table 22. Input local environment properties that can be used by the KafkaRead node		
Element name	**Type**	**Description**
LocalEnvironment.Destination.Kafka.Read.PartitionNumber	integer	The partition (in the topic) that contains the message.
LocalEnvironment.Destination.Kafka.Read.TopicName	string	The topic where the message was published.
LocalEnvironment.Destination.Kafka.Read.Offset	integer	The offset number in the partition for the message that was received.
LocalEnvironment.Destination.Kafka.Read.TimeoutInterval	integer	Timeout interval for reading the message from the Kafka topic.

The following table shows the data that is written by the KafkaRead node to the LocalEnvironment after reading a message:

Table 23. Data written by the KafkaRead node to the LocalEnvironment		
Element name	**Type**	**Description**
LocalEnvironment.Kafka.Read.partitionNumber	integer	The partition (in the topic) that contains the message.
LocalEnvironment.Kafka.Read.topicName	string	The topic where the message was published.
LocalEnvironment.Kafka.Read.offset	integer	The offset number in the partition for the message that was received.
LocalEnvironment.Kafka.Read.checksum	integer	The checksum of the message.
LocalEnvironment.Kafka.Read.key	string	The key (if any) that was associated with the received message. This field exists only if a key was associated with the message when it was published. The received key value is received as a string by the KafkaRead node.

You can use Kafka custom header properties to add metadata to Kafka messages for use during message processing. These properties are set in the LocalEnvironment, in a folder called `KafkaHeader`. For more information, see "Setting and retrieving Kafka custom header properties" on page 903.

Using Kafka nodes with IBM Event Streams

You can configure KafkaConsumer, KafkaRead, and KafkaProducer nodes to connect to the Event Streams service in IBM Cloud.

Before you begin

Read the following topics:

- "Processing Kafka messages" on page 892
- "Using Kafka with IBM App Connect Enterprise" on page 894

About this task

IBM Event Streams for IBM Cloud is a scalable, distributed, high-throughput message bus, which supports a number of client protocols including Kafka. You can use the KafkaConsumer, KafkaRead, and KafkaProducer nodes in IBM App Connect Enterprise to receive messages from and send messages to Event Streams.

Before you can connect to Event Streams, you must create a set of credentials, which the IBM App Connect Enterprise Kafka nodes can then use to make a connection. You can use either the **`mqsisetdbparms`** or **`mqsicredentials`** command to configure the credentials that the Kafka nodes use to authenticate to Event Streams.

To enable the Kafka nodes to authenticate by using the username and password, you must set the **`Security protocol`** property on the node to SASL_SSL.

For more information about configuring security credentials for connecting to Kafka, see "Configuring security credentials for connecting to Kafka" on page 904.

Procedure

Follow these steps to configure a connection to IBM Event Streams:

1. Create a set of credentials in Event Streams, which the Kafka nodes use for the connection.
2. In Event Streams, view the credentials and make a note of the list of servers in the **`kafka_brokers_sasl`** property.

   You use this list of servers to populate the **`Bootstrap servers`** property on the KafkaConsumer, KafkaRead, and KafkaProducer nodes when you are creating your message flow.

   You will use the values in the **`User`** and **`Password`** fields to configure the security credentials that IBM App Connect Enterprise uses to connect to Event Streams.

3. Use either the **mqsisetdbparms** or **mqsicredentials** command to associate a username and password with a connection to Event Streams:

   - Configure security credentials by using the **mqsisetdbparms** command, specifying the required username (**-u**), password (**-p**), and resource name (**-n**). The resource name is in the form `kafka::` followed by the name of the security identity that is specified on the Kafka node; for example, `kafka::myKafkaSecId`. Alternatively, you can use the default security identity, by specifying a resource name of `kafka::KAFKA` or `kafka::KAFKA::integrationServerName`.

     The following example shows how to specify a username, password, and named Kafka security identity:

     ```
     mqsisetdbparms -w workDir -n kafka::myKafkaSecId -u myUsername -p myPassword
     ```

     The following example hows how to specify a username and password, and specifies that the default Kafka security identity for the integration server will be used for connecting to Event Streams:

     ```
     mqsisetdbparms -w workDir -n kafka::KAFKA::myIntegrationServer1 -u myUsername -
     p myPassword
     ```

     For more information, see command.
   - Configure security credentials by using the **mqsicredentials** command, specifying the username (**--username**), password (**--password**), credential type (**--credential-type**), and credential name (**--credential-name**). Specify the credential type as `kafka`, and specify a credential name that matches the value of the **Security identifier** parameter specified in the node. Alternatively, you can use the default security identity specified by the **--set-as-default** parameter on the **mqsicredentials** command.

     The following example shows how to specify a username, password, and named Kafka security identity:

     ```
     mqsicredentials --create --work-dir workDir --credential-type kafka --credential-name
      myKafkaSecId --username myUsername --password myPassword
     ```

     The following example shows how to specify a username and password, and specifies that the default Kafka security identity for the integration server will be used for connecting to Event Streams:

     ```
     mqsicredentials --create --work-dir workDir --credential-type kafka --credential-name
      myKafkaSecId --username myUsername --password myPassword
     ```

     ```
     mqsicredentials --set-as-default --work-dir workDir --credential-type kafka --credential-
     name myKafkaSecId
     ```

     For more information, see command.
4. On the **Security** tab of the Kafka nodes, set the **Security protocol** property to SASL_SSL, and set the **SSL protocol** property to `TLSv1.2`.

   If the username and password that are to be used for connecting to Event Streams were configured by using the **mqsicredentials** command, specify the **Security identifier** property on the Kafka node, which will be used to access those credentials in the App Connect Enterprise vault.

What to do next

You can use the command or the command to show information about the credentials that are being used for connecting to Event Streams.

For more information about the Kafka nodes, see the following topics:

- node
- node

- node

Processing TCP/IP messages

You can use IBM App Connect Enterprise to connect to applications that use raw TCP/IP sockets for transferring data.

About this task

If you have existing applications that use raw TCP/IP sockets for transferring data, you can use the IBM App Connect Enterprise TCP/IP nodes to connect to the applications, without needing to enable them for IBM MQ.

IBM App Connect Enterprise implements access to the TCP/IP input and output streams through the following nodes:

- node
- node
- node
- node
- node
- node

For information about how to use the TCP/IP support, see “Working with TCP/IP” on page 923.

The following topics contain information that you need to understand before you can use TCP/IP in an IBM App Connect Enterprise application:

- “TCP/IP data transfer” on page 911
- “TCP/IP nodes” on page 914
- “Connection management” on page 917
- “Configuring TCP/IP client nodes to use SSL” on page 2625
- “Scenarios for IBM App Connect Enterprise and TCP/IP” on page 919

TCP/IP data transfer

You can use IBM App Connect Enterprise to connect to applications that use raw TCP/IP sockets for transferring data.

TCP/IP sockets provide a simple way of connecting computer programs together, and this type of interface is commonly added to existing stand-alone applications. TCP/IP provides a mechanism for transferring data between two applications, which can be running on different computers. The transfer of data is bidirectional; provided that the TCP/IP connection is maintained and no data is lost, the sequence of the data is kept. A significant advantage of using TCP/IP directly is that it is quick and simple to configure, which makes it a useful mechanism for processes that do not require message persistence (for example, monitoring).

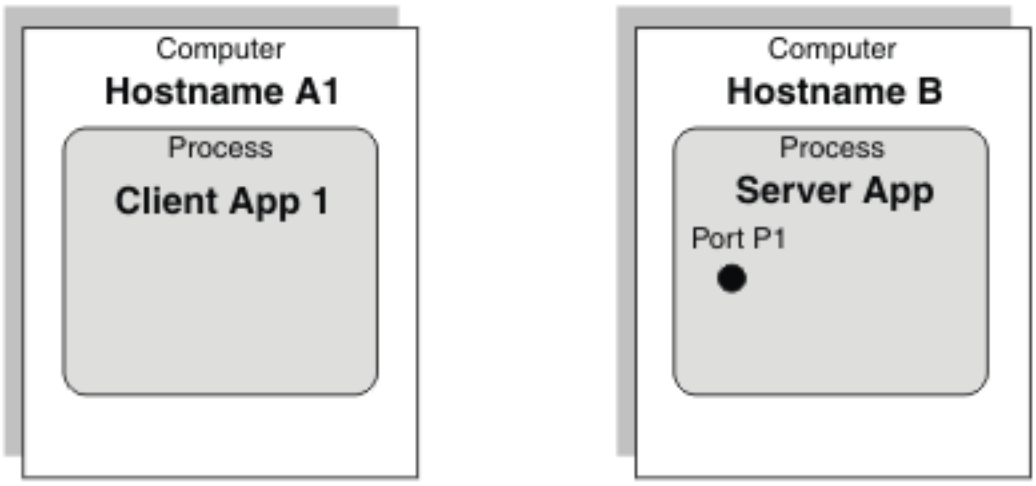
However, the use of TCP/IP sockets for transferring information between programs does have some limitations:

- It is non-transactional
- It is not persistent (the data is written to an in-memory buffer between the sender and receiver)
- It has no built-in security
- It provides no standard way of signaling the start and end of a message

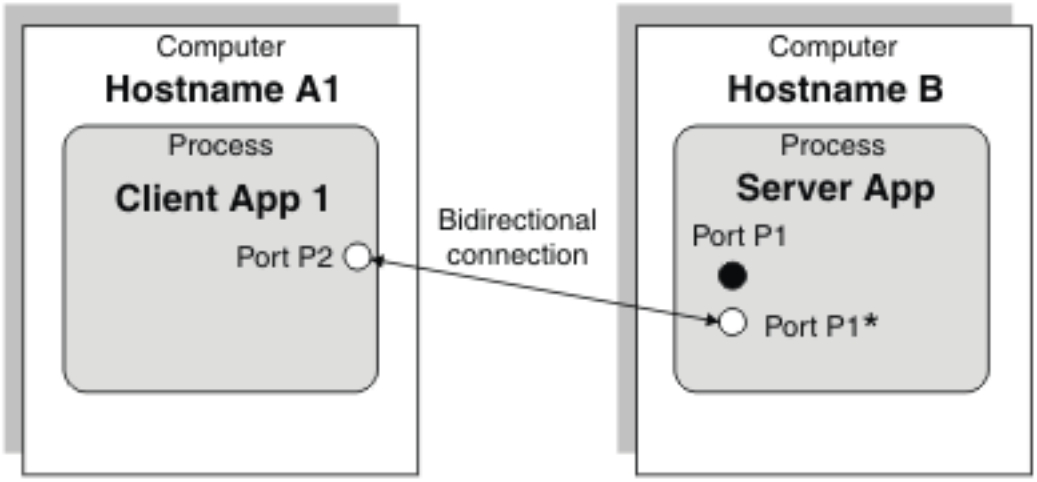
For these reasons, it can be preferable to use a transport mechanism like IBM MQ, which has none of these limitations. However, if you have existing applications that use raw TCP/IP sockets for transferring data, you can use the IBM App Connect Enterprise TCPIP nodes to connect to the applications without

needing to enable them for WebSphere MQ, so that you can develop an IBM App Connect Enterprise solution quickly.

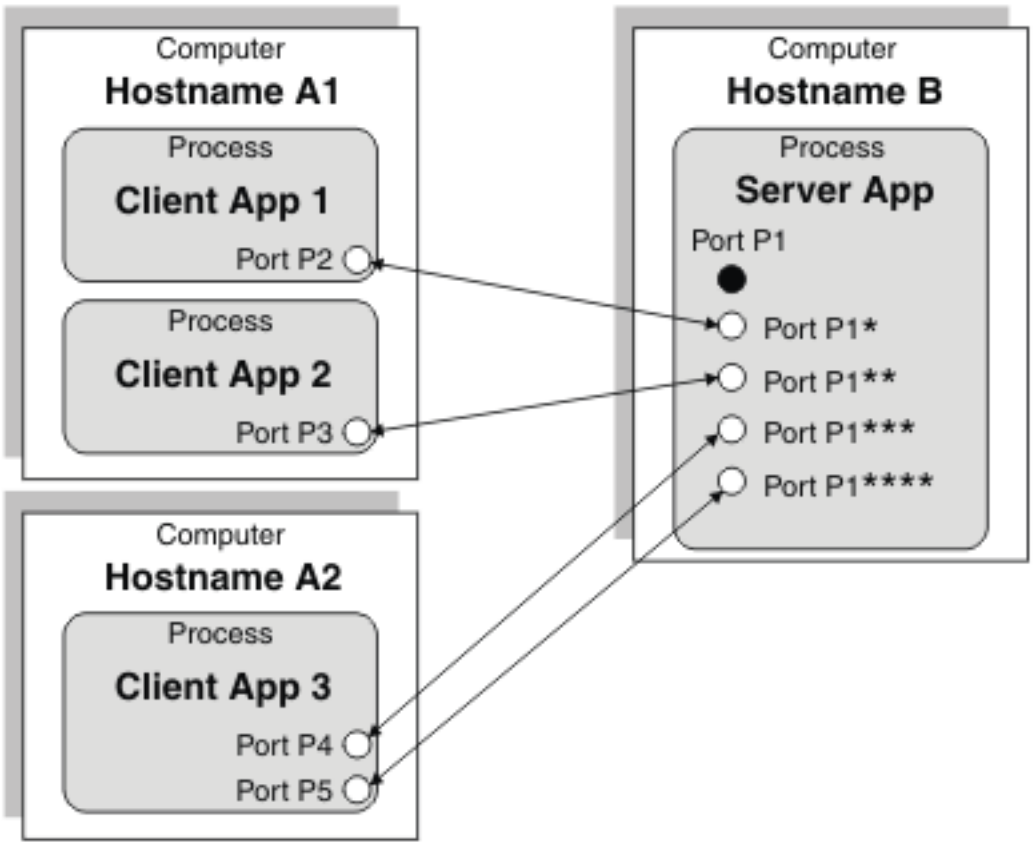
A TCP/IP connection between two applications has a client end and a server end, which means that one application acts as a server and the other as a client. The terms client and server refer only to the mechanism used to establish a connection; they do not refer to the pattern of data exchange. When the connection has been established, both client and server can perform the same operations and can both send and receive data. The following diagram illustrates the locations of client and server applications:



1. The server application listens on a local port (on the computer that is running the application) for requests for connections to be made by a client application.
2. The client application requests a connection from the server port, which the server then accepts.
3. When the server accepts the request, a port is created on the client computer and is connected to the server port.
4. A socket is created on both ends of the connection, and the details of the connection are encapsulated by the socket.
5. The server port remains available to listen for further connection requests:



The server can accept more connections from other client applications. These connections can be in the same process, in a different process on the same computer, or on a different computer:



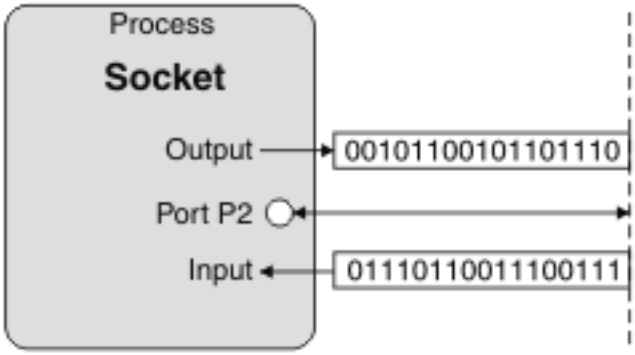
Only one server application can exist, but any number of different client processes can connect to the server application. Any of these applications (client or server) can be multithreaded, which enables them to use multiple connections.

When the connection has been established, two data streams exist: one for inbound data and another for outbound data:

The client and server ends of the connection are identical and both can perform the same operations. The only difference between them is that the output stream of the client is the input stream of the server, and the input stream of the client is the output stream of the server.

The two streams of data are independent and can be accessed simultaneously from both ends. The client does not need to send data before the server.

The example illustrated in the previous diagram can be simplified in the following way, showing that the client and server have access to a socket that has an input stream and an output stream:



TCP/IP nodes

IBM App Connect Enterprise implements access to the TCP/IP input and output streams through a series of message flow nodes.

Two sets of TCP/IP message flow nodes exist: TCPIPServer nodes and TCPIPClient nodes. Both sets have identical function in terms of accessing the data streams; however, one set uses client connections and the other set uses server connections. As a result, the nodes establish the connections in different ways but they use the streams in the same way when the connections have been established.

The main difference between the properties of the nodes is that the TCPIPServer nodes do not allow the host name to be changed (because it must always be `localhost`). All TCPIPServer nodes that use the same port must be in the same integration server because the port is tied to the running process. TCPIPClient nodes on the same port can be used in different integration servers, but client connections cannot be shared because the client connections are tied to a particular integration server, which maps to a process. Within the two sets of nodes (TCPIPClient and TCPIPServer), are three types of node:

- TCPIPServerInput and TCPIPClientInput
- TCPIPServerReceive and TCPIPClientReceive
- TCPIPServerOutput and TCPIPClientOutput

The input and receive nodes access the input stream to retrieve data, and the output nodes access the output stream to send data. No single node can access both streams at the same time. To access both streams simultaneously, you must connect multiple nodes in a message flow.

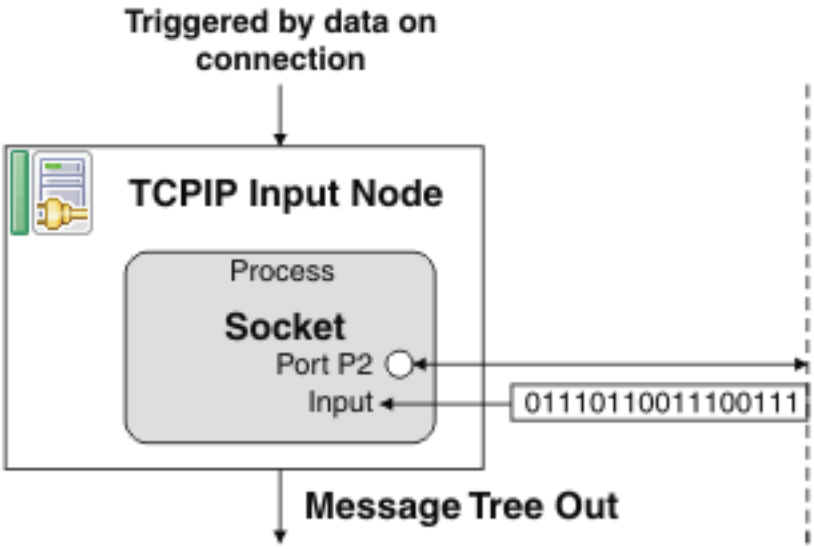
Policies for TCPIPServer and TCPIPClient nodes must be predefined by an administrator. TCP/IP policies must be defined before a message flow that uses them is deployed, even in a test environment.

Input nodes

The input node allows access to a connection's input stream. The node is triggered by the arrival of data in the stream and starts processing the message flow. The input node controls thread and transaction management. The TCP/IP nodes are not transactional in the way that they interact with TCP/IP, but other nodes in the same flow can be transactional (for example, WebSphere MQ nodes). The input node does not create a thread for every connection being used, but waits for two requirements to be met:

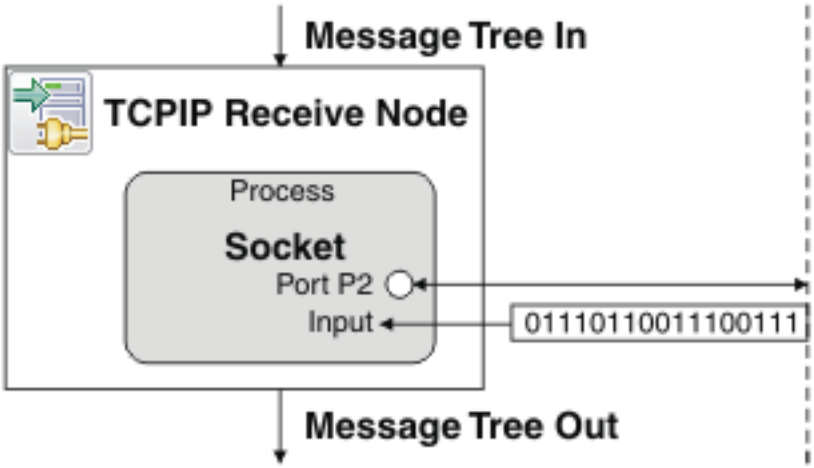
- A connection is available that still has an open input stream
- Data is available on the input stream (at least 1 byte)

For example, 1,000 TCP/IP connections can be handled by one input node that has only one additional instance. This situation is possible because the node does not poll the connections, but is triggered when the specified conditions are met.



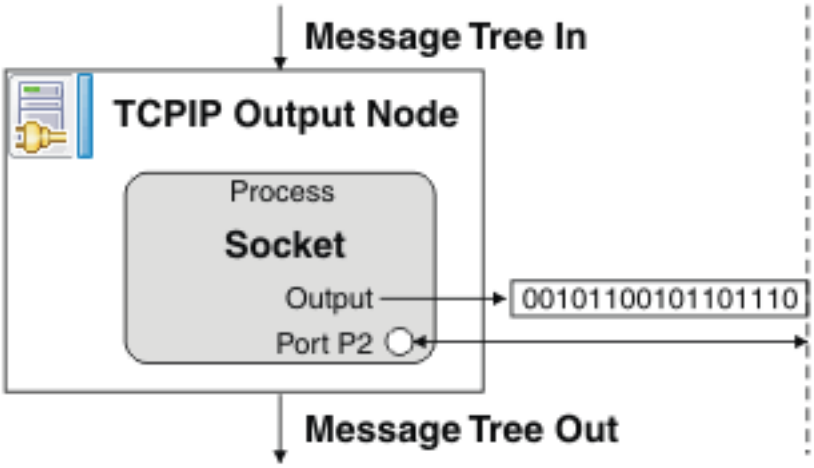
Receive nodes

The receive node is triggered to read data from a connection when a message arrives on its In terminal. It waits for data to arrive, then sends it to the Out terminal. You can configure the receive node to use a particular connection (by specifying a connection's ID) or to use any available connection. If the node is configured to use any available connection, it receives data from the first connection that has data available.



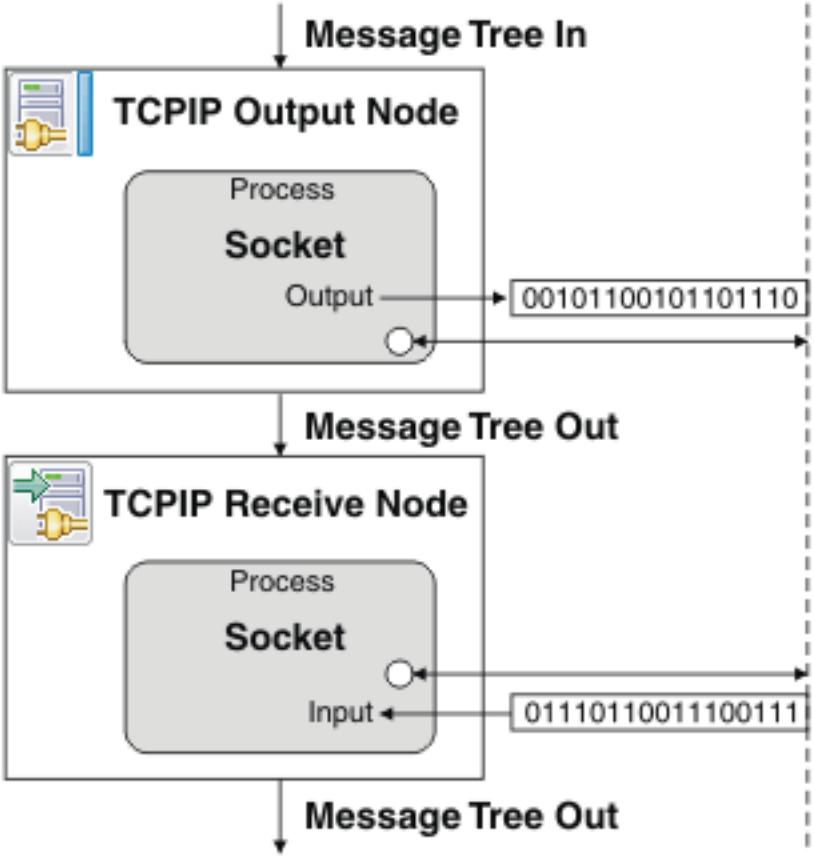
Output nodes

The output node sends data to a connection. It is triggered by a message arriving on its In terminal, then it sends the data contained in the message to the stream. The same message that is received in the node is sent to the Out terminal.



Combining nodes

The six client and server nodes can be combined to provide more complex operations. For example, an output node followed by a receive node enables a synchronous request of data:



If the message flows used are single threaded and only one connection ever exists, the sequence of nodes requires no further configuration. Two additional mechanisms are included to enable multithreading and multiple connections:

- A connection ID to ensure that the same connection is used by multiple nodes
- The ability to reserve connections so that they can be accessed only when the ID is specified

One connection in multiple nodes

Every connection has a unique identifier assigned to it when it is created. Whenever a node uses a connection, the ID that is used is written to the local environment. Any nodes that use it later in the flow can access the same connection by specifying the ID; the receive and output nodes find the ID by searching in a specified location in the local environment. By default, the location in which a node writes its connection details is different from the location in which the next node looks to see if there is an ID to use. The nodes can be configured to use the ID that was sent by a previous node. For example, the combination of the output and receive nodes shown in "Combining nodes" on page 916 can be configured so that the receive node uses the WrittenDestination data from the preceding output node.

The use of the ID enables a series of nodes to access the same connection, but does not prevent two message flow threads accessing the same connection. When a connection is used for the first time, it can be reserved so that no other nodes can access it unless they know the ID. For example, the combination of the output and receive nodes shown in "Combining nodes" on page 916 reserves the connection so that no other threads can access it before the receive node uses it. By default, the receive node then releases the connection when it has finished.

The ability to reserve connections (and access them by specifying the correct ID) enables you to build up complex interactions with TCP/IP connections that span whole flows and even multiple message flows. As a result, the TCP/IP interactions can be used with other asynchronous transport mechanisms like WebSphere MQ.

Reserved connections must be released at some time, otherwise they remain unavailable indefinitely. For more information about reserved and available connections, see "Connection management" on page 917.

Correlating replies across flows

You can use the TCP/IP nodes in asynchronous patterns, in which data is sent out through a TCP/IP output node and received back through a TCP/IP input node. The spanning of two message flows causes any state in the first flow to be lost and therefore inaccessible from the second flow. The TCP/IP nodes enable you to store some reply details on a connection, and these details are then available for the input node to use when a new event arrives on the same connection. By default, this data is taken from the local environment, but you can configure the nodes to take the data from any location, including the Correlid field and message IDs in IBM MQ headers.

Connection management

TCP/IP connections are requested by the client connection manager and accepted by the server connection manager.

The integration server process contains the connection manager, which makes the connections. Only one integration server can have TCPIP server nodes using a specific port at any one time; deployment to a second integration server causes a deployment error. TCPIP client nodes can be deployed to different integration servers, but each integration server has its own pool of connections, and therefore its own minimum and maximum number of connections.

TCPIP nodes do not directly create or manage any TCP/IP connections, but acquire them from the connection manager's internal pool. For example, two output nodes that use the same connection details share the same connection manager. The TCPIP nodes can define the connection details to be used by specifying one of the following values:

- Host name and port
- Name of a policy

If a host name and port are specified, the TCPIP node uses these values when it requests connections. If a policy is specified, the node obtains the values for the port and host name from the values that are defined in the policy. The connection manager supports other configurable parameters in addition to the host name and port, and you can define all of these values when you are using a policy. When the host name and port are specified on the node, the connection manager obtains the rest of the required values

from the default policy. However, if a policy is defined that is using the host name and port number, the values from that policy are used.

The connection manager is created when the first TCPIP node that requires connections from it is deployed. The connection manager is deleted when the last remaining node that uses it is removed from the integration server (which means that the connection manager is no longer being used by any deployed nodes). For example, this process can occur when existing flows are redeployed, because redeployment involves deleting all existing nodes before creating them again.

The connection manager applies any changes that are made to a TCPIP Client policy without requiring a restart of integration servers. The connection manager applies the TCPIP policy changes to all of the message flows that use that service. Applying TCPIP policy changes implies restarting the TCPIP connection managers, so all flows that use policies to specify TCPIP parameters can be expected to pick up new TCPIP connections.

The connection manager updates the affected TCPIP nodes only after every affected message flow has processed the current message. When an affected message flow has processed the current message, that message flow is locked until the updates to all affected message flows are complete. This process is to ensure that each node that is contained in the same integration server uses the same value for the modified policy. For this reason, there might be a delay of several seconds between applying the update to the policy, and the properties that are used by the message flow being updated.

You can view the statistics associated with TCPIP connections by using the **mqsireportproperties** command. For example:

```
mqsireportproperties INODE -e default -o TCPIP -r
```

Server connections

The server connection manager starts listening for server connections when it starts, and it keeps accepting connections until the maximum number of connections (as specified in the policy) is reached. Any attempts to make connections after this point are refused. TCP/IP servers do not create connections; they accept connection requests only from other applications. As a result, you cannot force the creation of connections within a message flow.

Connections to servers can be started, stopped, or quiesced. If you quiesce a connection, you must set a timeout value, or else quiescing continues forever. You can administer the connections by using the IBM Integration API, or the **mqsichangeproperties** command.

Client connections

The client connection manager starts and keeps making client connections until the minimum number of connections (as defined in the policy) is reached. By default, the minimum number of connections is zero, which means that no connections are made. Whenever the number of connections drops below the minimum value, the connection manager starts creating more client connections. The TCPIP client output and receive nodes initiate the creation of new client connections whenever none are available for them to use, unless the maximum number (as defined in the policy) has been reached.

Reserving and releasing connections

Each connection has an input stream and an output stream, both of which have two main states within the connection manager: available and reserved.

When a TCPIP node requests a connection for input or output, without specifying the ID of a particular connection, it is given any available connection on the required stream. If no connections are available, and if the node is a client node, a new connection is made, but only if the maximum number of connections has not yet been reached. Any connection in the available state can be used by only one node at a time, but when a node has finished using it, any other node (from any flow or thread) can access it.

You can restrict access to a stream on a connection by reserving the connection. When a connection is in the reserved state, no other node can access the stream without specifying the ID of the connection. For

example, an input node can request an available connection, and, when it has finished reading the data, put the stream into the reserved state. While the stream is in the reserved state, no input node (including the node that put the stream into the reserved state) can access it because input nodes can access only available streams. The only nodes that can access the stream must have the connection ID, which is written to the outgoing local environment when the data is passed down the message flow. As a result, receive nodes can read more data on the same connection, but only if the receive node is configured to use the ID from the local environment of the input node.

When a connection is reserved, ownership of the connection is given to a current thread of processing. This processing can span separate message flows, if required.

The reserve mechanism provides the following options:

- Leave unchanged
- Reserve
- Release
- Reserve and release at the end of the flow

For all nodes, the stream is left available (not reserved) by default. For many types of processing you can leave this default unchanged; for example, when you are moving data from an input stream to a file. The main purpose of reserving a stream is to connect a series of nodes to give complex processing on a stream in an ordered, controlled, synchronous sequence.

You can use the `Reserve and release at end of flow` option to reserve a connection and to ensure that the connection's stream is released when one iteration of the message flow has finished processing (including any error conditions that might occur).

If you require the processing to span multiple message flows (for example, for asynchronous request and reply), you must reserve a stream without releasing it at the end of the message flow.

A disadvantage of reserving a stream between message flows is the potential for a stream never to be released. To avoid this problem, set an expiry time on the connection so that it is closed after a specified period of inactivity.

Another benefit of reserving an input stream is that the connection cannot be closed until it is either released or expired (even if an end application closes its end of the connection), which is useful when the end of the stream is being used to delimit messages in the stream.

File descriptors

If your IBM App Connect Enterprise application is running on Sun Solaris 10 on SPARC, you might need to increase the number of file descriptors. The following error in the `syslog` indicates that additional file descriptors are required:

```
Failed to create a client connection using hostname: '', port: ''. Reason: 'Invalid argument'
```

You can also try the following two methods to resolve the error:

- Change the *MQSIJVERBOSE* environment variable, for example:

```
export MQSIJVERBOSE=-Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.PollSelectorProvider
```

- Change the limit of maximum file handles to *value* instead of *RLIM64_INFINITY*

Scenarios for IBM App Connect Enterprise and TCP/IP

Two example scenarios show how you might use TCP/IP and IBM App Connect Enterprise as part of a business solution.

- “Scenarios: TCP/IP” on page 920
- “Scenarios: IBM App Connect Enterprise using TCP/IP” on page 922

Scenarios: TCP/IP

Two scenarios illustrate how TCP/IP might be used as part of a business solution.

- "Expense submission" on page 920
- "Price-change notification" on page 921.

Expense submission

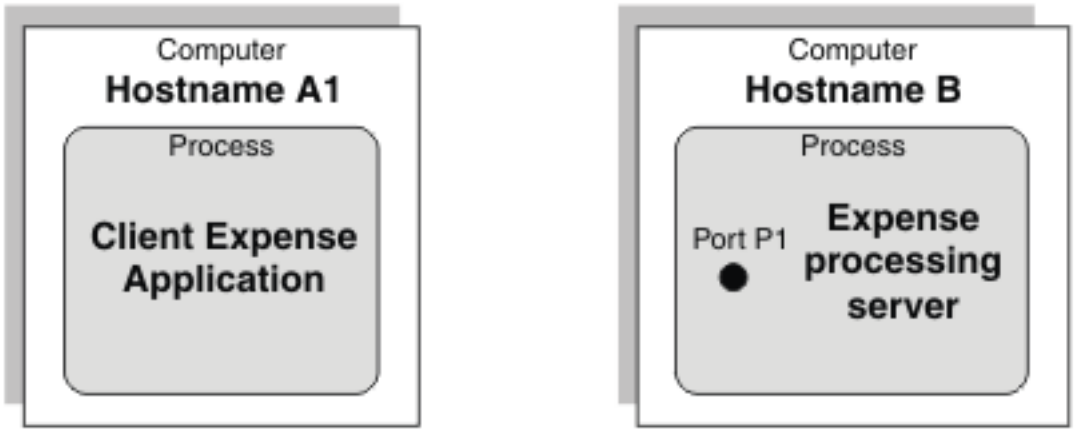
Scenario:

A company has an expense-submission system based on a central expense-processing application, which receives completed expense forms from end users. The users complete the forms by using a local application, which stores the form until it is completed. When it has been completed, the form is transferred to the central system where it is processed. Any further notifications are sent to the user by email.

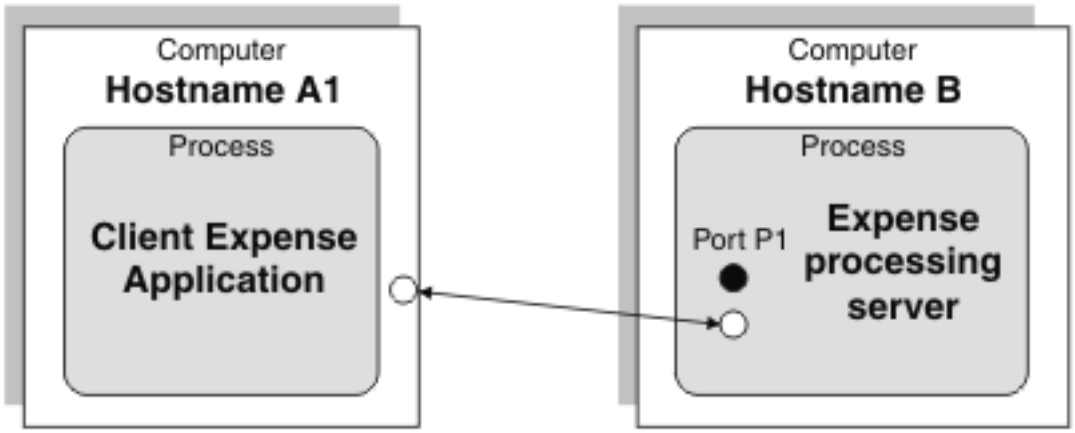
How TCP/IP is used:

One client application exists for each end user, and one central application processes all the expense forms. Each client application connects to a TCP/IP server port on the server application and sends the expense form in a fixed-field-size structure similar to a COBOL structure. The flow of processing is:

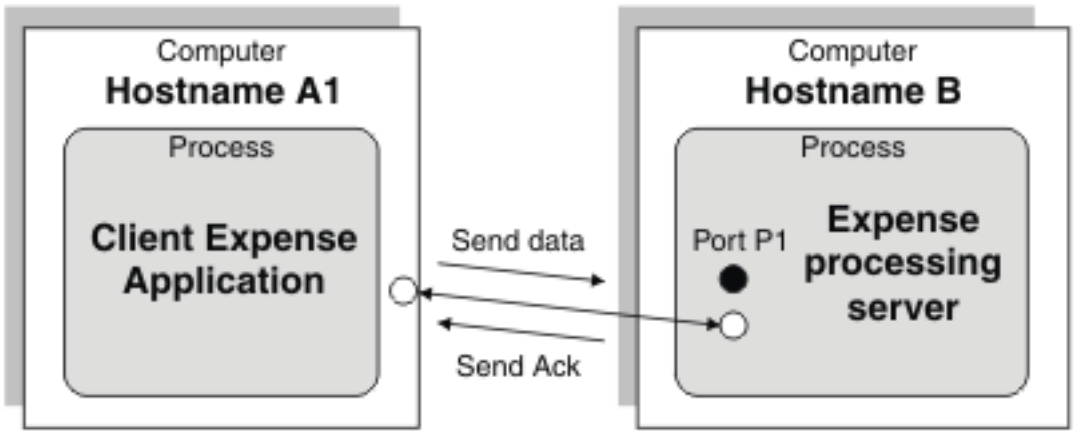
1. The user enters information into the form in the client expense application.



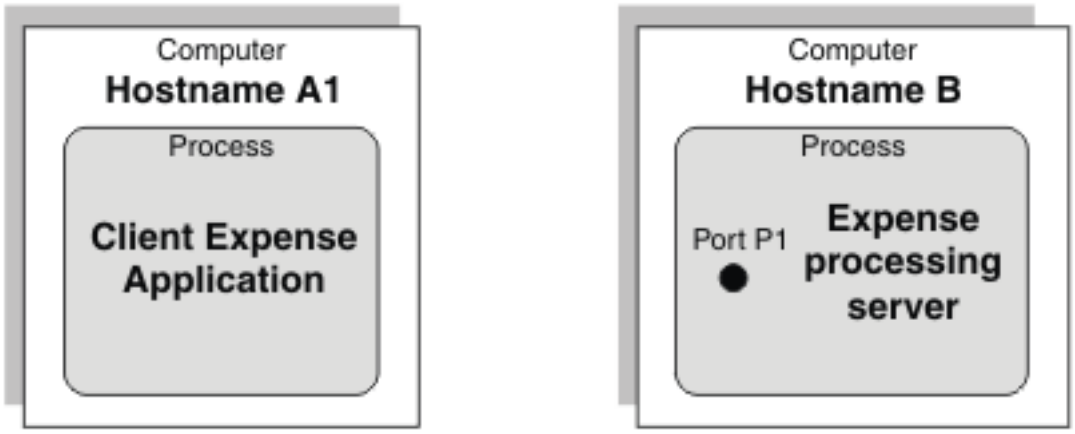
2. The user submits the completed form and the client application connects to the server.



3. The client application sends the data to the server and receives an acknowledgment.



4. The connection is closed by the client.



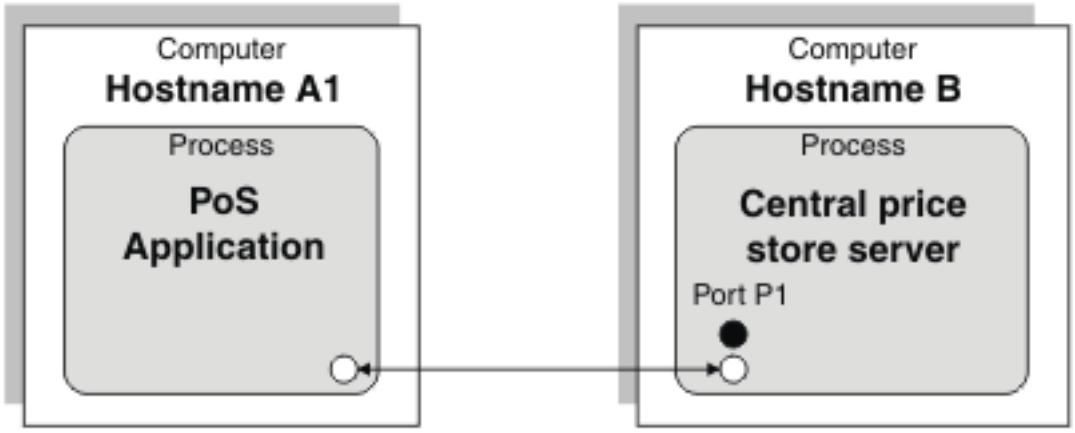
Price-change notification

Scenario:

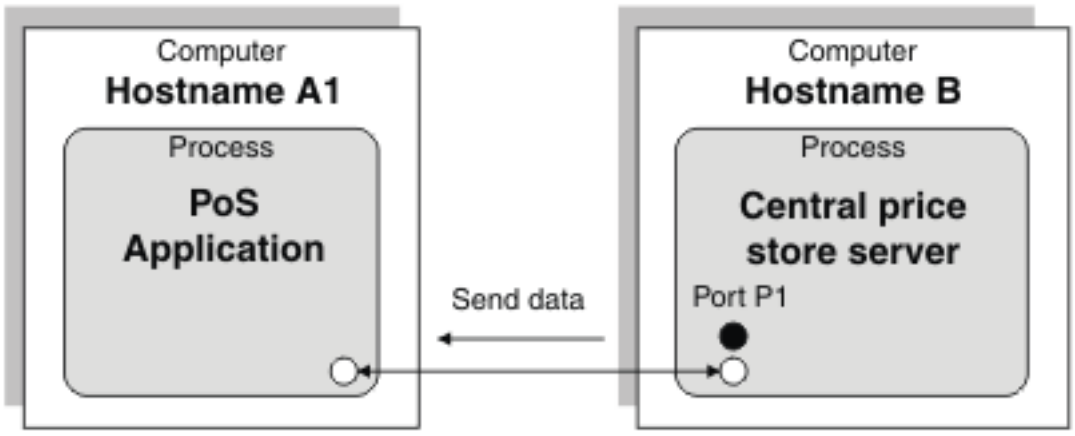
A company has a central server, which stores the catalog price of everything that the company sells. This information is required by all Point of Sale (PoS) terminals in all the stores, and each PoS terminal must be notified when any price changes. The PoS terminals connect to the central server and wait for any process changes. The server sends any process changes to all connected client applications.

How TCP/IP is used:

1. When the PoS application starts, it connects to the central server.



2. Whenever the server has a new price, it publishes it to all the connected clients.



3. The PoS stays connected until it shuts down.

See “Scenarios: IBM App Connect Enterprise using TCP/IP” on page 922 for an example of how these scenarios can be modified to use IBM App Connect Enterprise.

Scenarios: IBM App Connect Enterprise using TCP/IP

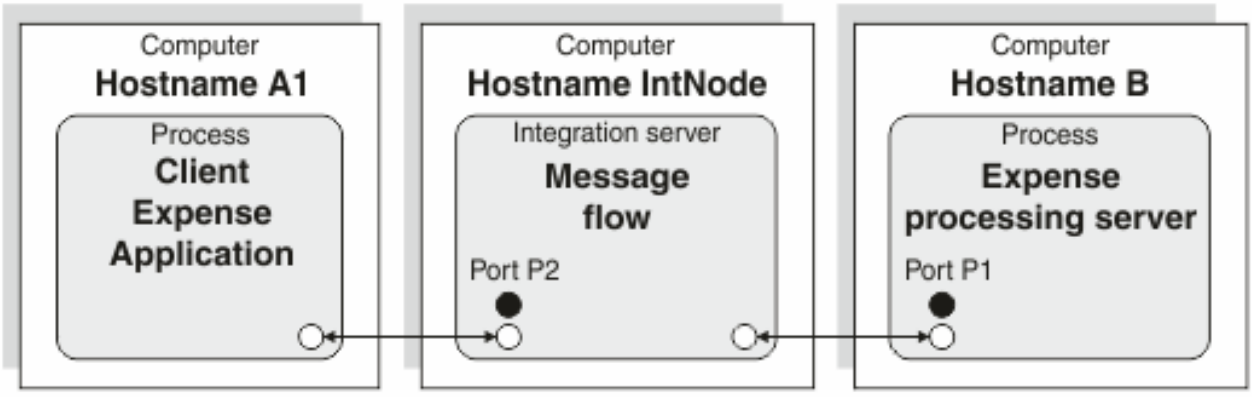
IBM App Connect Enterprise can be added to systems that use TCP/IP for transport, to generate a more flexible architecture for communication between components.

The scenarios in the “Scenarios: TCP/IP” on page 920 topic show how systems can be created to use TCP/IP as a transport mechanism. The following sections show how IBM App Connect Enterprise can be added to those systems to generate a more flexible architecture for communication between components:

- “Expense submission” on page 922 illustrates TCP/IP to TCP/IP routing
- “Price-change notification” on page 922 illustrates routing and transformation to other formats

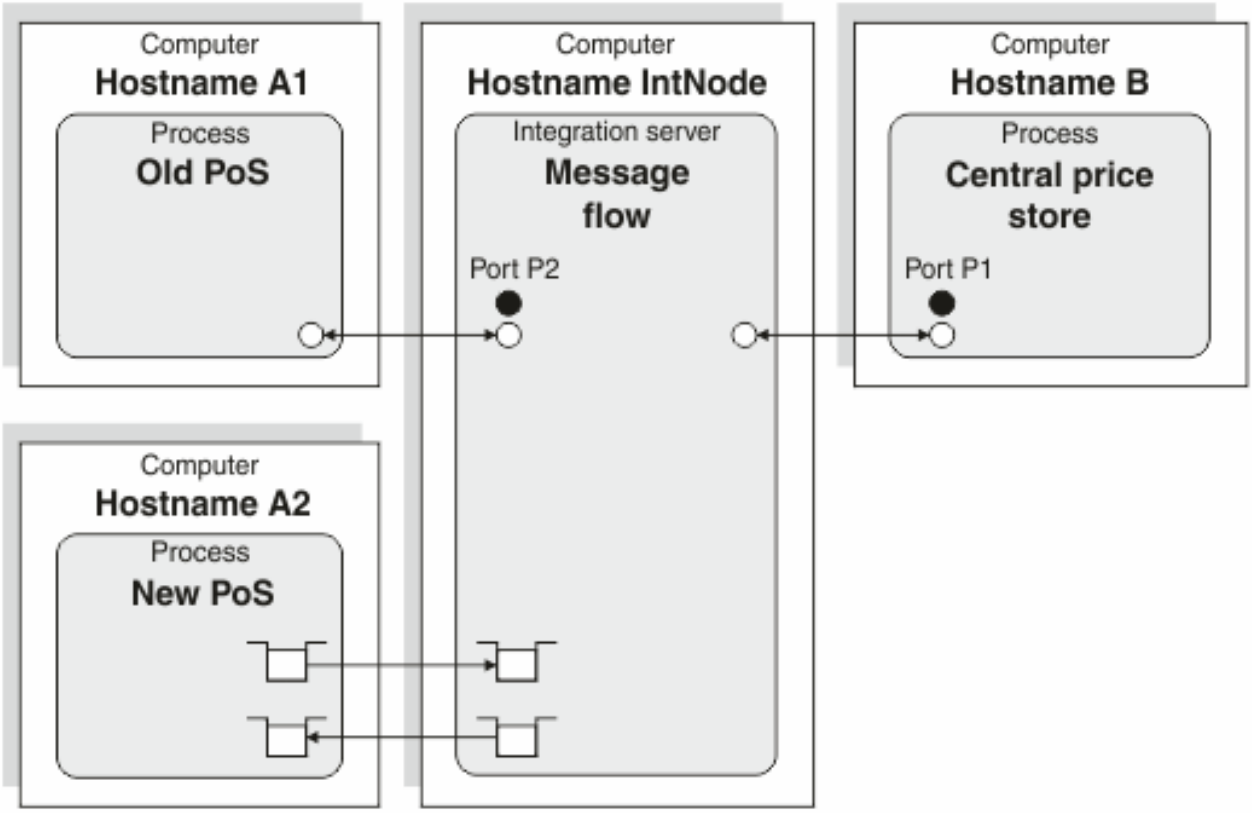
Expense submission

The expense submission scenario shown in the “Scenarios: TCP/IP” on page 920 topic requires a direct connection from the client applications to the end server. With that model it is difficult to add new consumers of the expense submission information, and it is also difficult to change the end application that processes them. However, by adding IBM App Connect Enterprise as an intermediary router, the two systems can be separated without any changes to their interfaces, as shown in the following diagram:



Price-change notification

The price-change notification scenario shown in the “Scenarios: TCP/IP” on page 920 topic can be modified to use an integration node for routing and transformation. It could also allow support for other protocols like IBM MQ, which would allow new applications to be written to different interfaces without the need for changing the current client or server applications:



Working with TCP/IP

You can use IBM App Connect Enterprise TCPIP nodes and TCP/IP policies to perform various tasks.

About this task

- “Sending and receiving data on the same TCP/IP client connection, closing input and output streams (synchronous)” on page 937.
- “Configuring TCP/IP client nodes to use SSL” on page 2625

Transferring XML data from a TCP/IP server socket to an IBM MQ queue
Transfer XML data from a TCP/IP server socket to an IBM MQ queue, by creating a message flow with TCPIPServerInput and MQOutput nodes.

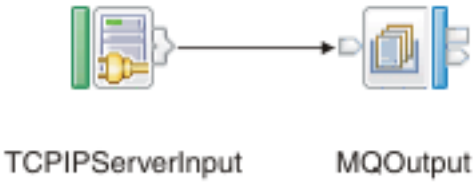
About this task

Scenario: A client application opens a TCP/IP socket and sends an XML document. The end of the document is signalled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the XML document and write it to an IBM MQ queue:

Procedure

1. Create a message flow called TCPIP_Task1 with a TCPIPServerInput node and an MQOutput node.
   For more information about how to do this, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a) On the **Basic** tab, set the `Connection details` property to 14141.
   b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK1.OUT1`.
5. Save the message flow.

Transferring binary (CWF) data from a TCP/IP server socket to a flat file
Transfer binary Custom Wire Format (CWF) data from a TCP/IP server socket to a flat file, by the use of a message set and a message flow with TCPIPServerInput and FileOutput nodes.

About this task

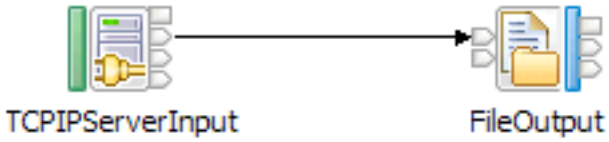
Scenario: A client application opens a TCP/IP socket and sends a binary (CWF) document. The end of the document is signaled by the closure of the client connection.

Instructions: The following steps describe how to write a message flow that can receive the binary document and write it to a flat file. Each message is written to a separate file, the name of which is based on the ID of the connection.

Procedure

1. Create a message set called Task2_MsgSet.
   For more information, see “Message Sets: Creating a message set” on page 2250.
2. Create a message flow called TCPIP_Task2 with a TCPIPServerInput node and a FileOutput node.
   For more information, see “Creating a message flow” on page 574.

3. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the FileOutput node.



4. Set the following properties of the TCPIPServerInput node:
   a) On the **Basic** tab, set the `Connection details` property to 14142.
   b) On the **Input Message Parsing** tab, set the following properties:
      - Set the `Message domain` property to `MRM`.
      - Set the `Message model` property to `Task2_MsgSet`.
      - Set the `Message` property to `Task2_MsgType`.
      - Set the `Physical format` property to `Binary1`.
5. Set the following properties of the FileOutput node:
   a) On the **Basic** tab, set the following properties:
      - Set the `Directory` property to `c:\temp\task2`.
      - Set the `File name or pattern` property to `Task2.out`.
   b) On the **Request** tab, set the `Request file name property location` property to `$LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
6. Save the message flow.
7. Create a project reference between the integration project and the message set project.
   For more information, see "Referencing resources in other libraries" on page 1976.

Receiving data on a TCP/IP server socket and sending data back to the same connection
Receive data on a TCP/IP server socket, then send the data to the same connection, by the use of a message flow with TCPIPServerInput and TCPIPServerOutput nodes.

About this task

Scenario: A client application opens a TCP/IP socket and sends an undefined document (of any format or size). The end of the document is signalled by the client closing the output stream (but not the connection), and waiting for the same data to be sent back.

Instructions: The following steps describe how to write a message flow that can receive the data and echo it back to the same connection:

Procedure

1. Create a message flow called TCPIP_Task3 with a TCPIPServerInput node and a TCPIPServerOutput node.
   For more information, see "Creating a message flow" on page 574.

2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the TCPIPServerOutput node.

3. Set the following properties of the TCPIPServerInput node:
   a) On the **Basic** tab, set the `Connection details` property to 14143.
   b) On the **Advanced** tab, set the `Input stream modification` property to `Reserve input stream and release at end of flow`.
4. Set the following properties of the TCPIPServerOutput node:
   a) On the **Basic** tab, set the `Connection details` property to 14143.
   b) On the **Request** tab, set the `ID location` property to `LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
   c) On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
5. Save the message flow.

Sending XML data from an IBM MQ queue to a TCP/IP client socket
Send XML data from an IBM MQ queue to a TCP/IP client socket, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

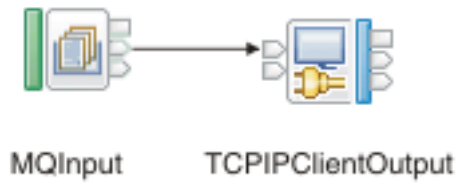
About this task

Scenario: A server application listens on a TCP/IP socket and waits for a TCP/IP client to connect and send data. The end of the document is signalled by the client closing the connection.

Instructions: The following steps describe how to write a message flow that can take a message from an IBM MQ queue, make the client connection, and send the data to the server application:

Procedure

1. Create a message flow called TCPIP_Task4 with an MQInput node and a TCPIPClientOutput node.
   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.



3. Set the following properties of the MQInput node:
   a) On the **Basic** tab, set the `Queue name` property to `TCPIP.TASK4.IN1`.
   b) On the **Input message parsing** tab, set the `Message domain` property to `XMLNSC`.
4. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to 14144.
   b) On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
5. Save the message flow.

Sending CWF data from a flat file to a TCP/IP client socket
Send Custom Wire Format (CWF) data from a flat file to a TCP/IP client socket, by the use of a message flow with FileInput and TCPIPClientOutput nodes.

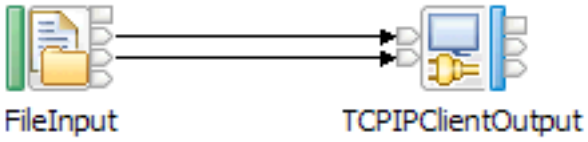
About this task

Scenario: An application writes 100-byte binary records into a flat file.

Instructions: The following steps describe how to open a new client TCP/IP connection and send the binary data with a binary termination character x'00FF'. When the whole file is finished, the client connection is closed:

Procedure

1. Create a message flow called TCPIP_Task5 with a FileInput node and a TCPIPClientOutput node.
   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the FileInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the End of Data terminal of the FileInput node to the Close terminal of the TCPIPClientOutput node.



4. Set the following properties of the FileInput node:
   a) On the **Basic** tab, set the `Input directory` property to `c:\temp\task5`.
   b) On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
5. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14145`.
   b) On the **Advanced** tab, set the `Close connection` property to `After data has been sent`.
   c) On the **Records and elements** tab, set the following properties:
      - Set the `Record definition` property to `Record is delimited data`.
      - Set the `Delimiter` property to `Custom delimiter (Hexadecimal)`.
      - Set the `Custom delimiter` property to `00FF`.
6. Save the message flow.

Sending data to a TCP/IP client connection and receiving data back on the same connection (synchronous)
Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (synchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

About this task

Scenario: An application sends synchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow that sends data out from a client connection and waits on the same connection for a reply to be returned. The request is synchronous in the same flow, because the TCPIPClientReceive node waits for data to be returned.

Procedure

1. Create a message flow called TCPIP_Task6 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node.

   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.

5. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK6.IN1`.
6. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14146`.
   b) On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
7. Set the following properties of the TCPIPClientReceive node:
   a) On the **Basic** tab, set the `Connection details` property to `14146`.
   b) On the **Advanced** tab, set the following properties:
      - Set the `Output stream modification` property to `Reserve output stream and release at end of flow`.
      - Set the `Input stream modification` property to `Reserve input stream and release at end of flow`.
   c) On the **Request** tab, set the `ID location` property to `$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id`.
   d) On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
8. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK6.OUT1`.
9. Save the message flow.

Sending data to a TCP/IP client connection and receiving data back on the same connection (asynchronous)

Send fixed-size data to a TCP/IP client connection and receive fixed-size data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

About this task

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows (the TCPIPClientInput does not wait for data to be returned on this connection, but instead monitors all available connections).

Procedure

1. Create a message flow called TCPIP_Task7 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientInput node, and an MQOutput node.

   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.

4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK7.IN1`.
5. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14147`.
   b) On the **Advanced** tab, set the `Output stream modification` property to `Reserve output stream`.
   c) On the **Records and elements** tab, set the following properties:
      - Set the `Record definition` property to `Fixed length`.
      - Set the `Length` property to `100`.
6. Set the following properties of the TCPIPClientInput node:
   a) On the **Basic** tab, set the `Connection details` property to `14147`.
   b) On the **Advanced** tab, set the `Output stream modification` property to `Release output stream and reset Reply ID`.
   c) On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Fixed length`.
      - Set the `Length` property to `100`.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK7.OUT1`.
8. Save the message flow.

Establishing a client session over a TCP/IP connection
Configure a TCPIPClientInput node to open a session over an existing TCP/IP connection, before any data is sent or received.

About this task

You can use the Open terminal of the TCPIPClientInput node to enable processing to start when a connection is opened, rather than when data first arrives. If the Open terminal is connected, an empty message is sent to the Open terminal when a connection is created. This message has the local environment set to the connection details.

The connection associated with the message is reserved from the general connection pool until propagation to the Open terminal has finished. However, the connection can be accessed using the connectionId specified in the local environment. Each connection that is created is sent to the Open terminal, including any connections that are created mid-flow by a TCPIPClientReceive node or TCPIPClientOutput node.

If the Open terminal is not attached, open events are automatically made available in the connection pool.

The following steps show how to configure a message flow that contains a TCPIPClientInput node, with the Open terminal configured to start processing when a TCP/IP connection is created.

Procedure

1. Create a message flow containing a TCPIPClientInput node, a Compute node, and a TCPIPClientOutput node.

   For information about how to do this, see “Creating a message flow” on page 574“Creating a message flow” on page 574.
2. Connect the Open terminal of the TCPIPClientInput node to the In terminal of the Compute node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the TCPIPClientOutput node.
4. On the TCPIPClientInput node, set the `Connection details` property (on the **Basic** tab) to 14143.
5. On the Compute node, set the ESQL property (on the **Basic** tab) to:

```
CREATE COMPUTE MODULE test_Compute1
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  CALL CopyEntireMessage();
  RETURN TRUE;
 END;

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

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

6. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to 14143.
   b) On the **Request** tab, set the `ID location` property to `LocalEnvironment/TCPIP/Input/ConnectionDetails/Id`.
7. Save the message flow.

Broadcasting data to all currently available connections
Broadcast data to all current connections, by the use of a message flow with MQInput and TCPIPServerOutput nodes.

About this task

Scenario: Several applications connect into the message flow and wait to be sent data.

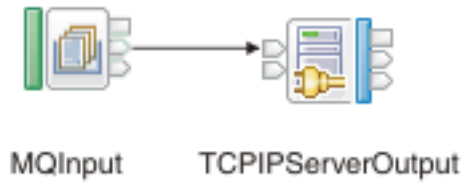
Instructions: The following steps describe how to send data to all the connected client applications:

Procedure

1. Create a message flow called TCPIP_Task8 with an MQInput node and a TCPIPServerOutput node.

   For more information, see “Creating a message flow” on page 574.

2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerOutput node.



3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK8.IN1`.
4. Set the following properties of the TCPIPServerOutput node:
   a) On the **Basic** tab, set the `Connection details` property to 14148.
   b) On the **Advanced** tab, set the `Send to:` property (in the **Broadcast options** group) to `All available connections`.
5. Save the message flow.

Configuring a server socket to receive XML data ending in a null character
Configure a server TCP/IP socket to receive XML data ending in a null character, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

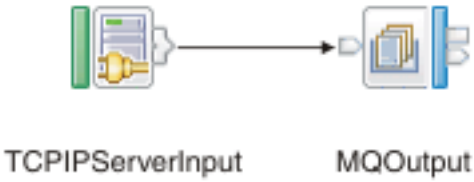
About this task

Scenario: A client application sends XML data that is delimited by a null character (hex code '00').

Instructions: The following steps describe how to break up the record based on the null character, then parse the data.

Procedure

1. Create a message flow called TCPIP_Task11 with a TCPIPServerInput node and an MQOutput node. For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a) On the **Basic** tab, set the `Connection details` property to 14151.
   b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
   c) On the **Records and elements** tab, set the following properties:
      - Set the `Record detection` property to `Delimited`.
      - Set the `Delimiter` property to `Custom delimiter`.
      - Set the `Custom delimiter` property to `00`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK11.IN1`.
5. Save the message flow.

Configuring a server socket to receive XML data and discover the end of a record (by using the message model)
Configure a server socket to receive XML data and use the message model to determine the end of a record, by the use of a message flow with TCPIPServerInput and MQOutput nodes.

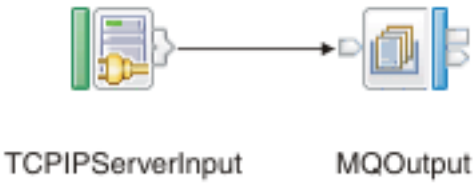
About this task

Scenario: A client application sends an XML document with no clear indication of the end of the record.

Instructions: The following steps show how to break up the record by the use of the XML parser to signal when the whole XML document as been received. The parser uses the end XML tag to signal the end of the message.

Procedure

1. Create a message flow called TCPIP_Task12 with a TCPIPServerInput node and an MQOutput node.
   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the TCPIPServerInput node to the In terminal of the MQOutput node.



3. Set the following properties of the TCPIPServerInput node:
   a) On the **Basic** tab, set the `Connection details` property to `14151`.
   b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
   c) On the **Records and elements** tab, set the `Record detection` property to `Parsed record sequence`.
4. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK12.IN1`.
5. Save the message flow.

Configuring a server output node to close all connections
Configure a server output node to close all connections on a specified port.

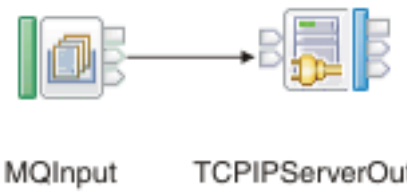
About this task

Scenario: A server output node is configured to close all connections on a specified port.

Instructions: The following steps show how to create a message flow that closes all TCP/IP connections on port 14153:

Procedure

1. Create a message flow called TCPIP_Task13 with an MQInput node and a TCPIPServerOutput node.
   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the MQInput node to the Close terminal of the TCPIPServerOutput node.



3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK13.IN1`.
4. Set the following properties of the TCPIPServerOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14153`.
   b) On the **Advanced** tab, set the `Send to:` property (in the **Broadcast options** group) to `All available connections`.
5. Save the message flow.

Configuring a client socket to store reply correlation details
Configure a client TCP/IP socket to store reply correlation details, by the use of a message flow with MQInput and TCPIPClientOutput nodes.

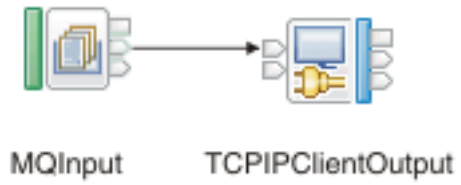
About this task

Scenario: A client TCP/IP socket is configured to store response returned on the input stream.

Instructions: The following steps show how to set the Reply ID on a connection, which can be used when a response is returned on the input stream:

Procedure

1. Create a message flow called TCPIP_Task14 with an MQInput node and a TCPIPClientOutput node.
   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.



3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK14.IN1`.
4. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14154`.
   b) On the **Request** tab, set the `Reply ID location` property to `$Root/MQMD/MsgId`.
5. Save the message flow.

Writing close connection details to a file
Configure a message flow to write details of a connection closure to a file, by the use of TCPIPServerInput, Compute, and FileOutput or FTEOutput nodes.

About this task

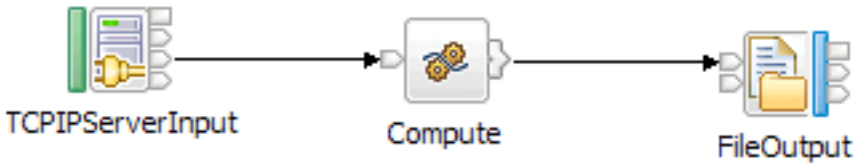
Scenario: A message flow writes close connection details to a file. The scenario uses the FileOutput node; the steps shown also apply to the FTEOutput node.

Instructions The following steps show how to configure a message flow to write details of the closure of any connection to a file:

Procedure

1. Create a message flow called TCPIP_Task15 with a TCPIPServerInput node, a Compute node, and a FileOutput node.
   For more information, see “Creating a message flow” on page 574.
2. Connect the Close terminal of the TCPIPServerInput node to the In terminal of the Compute node.

3. Connect the Out terminal of the Compute node to the In terminal of the FileOutput node.



4. On the TCPIPServerInput node, set the `Connection details` property (on the **Basic** tab) to 14155.
5. On the Compute node, set the ESQL property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks


CREATE COMPUTE MODULE TCPIP_Task15_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  Set OutputRoot.XMLNSC.CloseEvent = InputLocalEnvironment.TCPIP;
  RETURN TRUE;
 END;

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

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

6. Set the following properties of the FileOutput node.
   a) On the **Basic** tab, set the following properties:
      - Set the `Directory` property to `c:\temp\Task15`.
      - Set the `File name or pattern` property to `CloseEvents.txt`.
   b) On the **Records and elements** tab, set the `Record definition` property to `Record is unmodified data`.
7. Save the message flow.

Configuring a client node to dynamically call a port
Configure a client node to dynamically use a port and hostname that are set in the local environment, by the use of a message flow with MQInput, Compute, and TCPIPClientOutput nodes.

About this task

Scenario: A client node dynamically calls a port.

Instructions: The following steps show how to override the connection details specified on a client output node to dynamically use a port and hostname that are set in the local environment:

Procedure

1. Create a message flow called TCPIP_Task16 with an MQInput node, a Compute node, and a TCPIPClientOutput node.
   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the Compute node.

3. Connect the Out terminal of the Compute node to the In terminal of the TCPIPClientOutput node.

4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK16.IN1` .
5. On the Compute node, set the ESQL property (on the **Basic** tab) to:

```
BROKER SCHEMA Tasks
CREATE COMPUTE MODULE TCPIP_Task16_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  -- CALL CopyMessageHeaders();
  CALL CopyEntireMessage();
  set InputLocalEnvironment.Destination.TCPIP.Output.Hostname = 'localhost';
  set InputLocalEnvironment.Destination.TCPIP.Output.Port = 14156;
  RETURN TRUE;
 END;

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

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

6. On the TCPIPClientOutput node, set the `Connection details` property (on the **Basic** tab) to 9999.
7. Save the message flow.

Configuring a server receive node to wait for data on a specified port
Configure a TCPIPServerReceive node to block the message flow and wait for data to arrive on any connection.

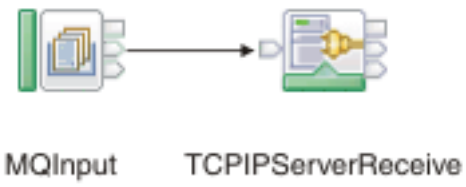
About this task

Scenario: A server receive node is configured to wait for data on a specified port.

Instructions: The following steps show how to configure a TCPIPServerReceive node to wait for data on port 14157:

Procedure

1. Create a message flow called TCPIP_Task17 with an MQInput node and a TCPIPServerReceive node. For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPServerReceive node.



3. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK17.IN1`.

4. On the TCPIPServerReceive node, set the `Connection details` property (on the **Basic** tab) to 14157.
5. Save the message flow.

Sending and receiving data through a TCP/IP client connection, delimiting the record by closing the output stream (asynchronous)
Send data through a TCP/IP client connection and receive data back on the same connection (asynchronously), by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientInput, and MQOutput nodes.

About this task

Scenario: An application sends asynchronous data between TCP/IP client connections.

Instructions: The following steps describe how to create a message flow to send data through a client connection and wait on the same connection for a reply to be returned. The request is performed asynchronously in two different flows; the TCPIPClientInput node does not wait for data to be returned on this connection, but monitors all available connections. The outgoing record is delimited by closing the output stream, and the reply message is delimited by the remote server closing the input stream. The connection is then completely closed by the node.

Procedure

1. Create a message flow called TCPIP_Task18 with an MQInput node, a TCPIPClientOutput, a TCPIPClientInput node, and an MQOutput node.
   For more information, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientInput node to the In terminal of the MQOutput node.

4. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK18.IN1`.
5. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to 14158.
   b) On the **Advanced** tab, select `Close output stream after a record has been sent`.
   c) On the **Records and elements** tab, set the `Record definition` property to `Record is unmodified data`.
6. Set the following properties of the TCPIPClientInput node:
   a) On the **Basic** tab, set the `Connection details` property to 14158.
   b) On the **Advanced** tab, set the `Close connection` property to `After data has been received`.
   c) On the **Records and elements** tab, set the `Record detection` property to `End of stream`.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK18.OUT1`.
8. Save the message flow.

Sending and receiving data on the same TCP/IP client connection, closing input and output streams (synchronous)

Send data through a TCP/IP client connection and wait on the same connection for a reply to be returned, by the use of a message flow with MQInput, TCPIPClientOutput, TCPIPClientReceive, and MQOutput nodes.

About this task

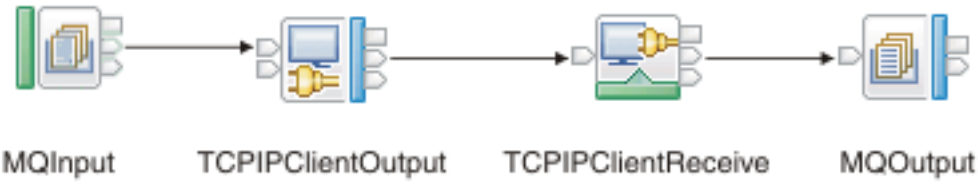
Scenario: An application sends synchronous data on the same TCP/IP client connection.

Instructions: The following steps describe how to create a message flow that sends out data through a client connection and waits on the same connection for a reply to be returned. The request is synchronous within the same flow, as a result of the TCPIPClientReceive node waiting for data to be returned. The outgoing message is delimited by closing the output stream, and the reply data is delimited by the remote application closing the input stream.

Procedure

1. Create a message flow called TCPIP_Task19 with an MQInput node, a TCPIPClientOutput node, a TCPIPClientReceive node, and an MQOutput node.

   For more information, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the TCPIPClientOutput node.
3. Connect the Out terminal of the TCPIPClientOutput node to the In terminal of the TCPIPClientReceive node.
4. Connect the Out terminal of the TCPIPClientReceive node to the In terminal of the MQOutput node.



5. On the MQInput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK19.IN1`.
6. Set the following properties of the TCPIPClientOutput node:
   a) On the **Basic** tab, set the `Connection details` property to `14159`.
   b) On the **Advanced** tab, set the following properties:
      - Select `Close output stream after a record has been sent`.
      - Set the `Input stream modification` property to `Reserve input stream and release at end of flow`. It is important to reserve the input stream so that it is not closed before the receive node processes the return data.
   c) On the **Records and elements** tab, set the `Record definition` property to `Record is Unmodified Data`.
7. Set the following properties of the TCPIPClientReceive node:
   a) On the **Basic** tab, set the `Connection details` property to `14159`.
   b) On the **Advanced** tab, set the `Close connection` property to `After data has been received`.
   c) On the **Request** tab, set the `ID location` property to `$LocalEnvironment/WrittenDestination/TCPIP/Output/ConnectionDetails[1]/Id`.
   d) On the **Records and elements** tab, set the `Record detection` property to `Connection closed`.
8. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `TCPIP.TASK19.OUT1`.
9. Save the message flow.

Processing email messages

You can configure the EmailOutput node to deliver an email from a message flow to an email server that supports Simple Mail Transfer Protocol (SMTP). You can also configure the EmailInput node to retrieve an email from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

About this task

The topics in the following sections describe how the EmailOutput and the EmailInput nodes work, and how to use them to send and receive email messages.

- "Sending emails" on page 938
- "Receiving emails" on page 947

Sending emails

You can use the EmailOutput node to send email messages, with or without attachments, to one or more recipients. The EmailOutput node delivers an email message from your message flow to an email server that supports Simple Mail Transfer Protocol (SMTP), which you specify.

The following topics describe how the EmailOutput node works and how to use it to send email messages.

- "Sending emails" on page 938
- "Sending an email" on page 939
- "Sending an email with an attachment" on page 940
- "Producing dynamic email messages" on page 940
- "Sending a MIME message" on page 944
- "Changing connection information for the EmailOutput node" on page 946

Receiving emails

You can use the EmailInput node to receive email messages, with or without attachments, from one or more recipients. The EmailInput node retrieves an email message from an email server that you specify, which supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

The following topics describe how the EmailInput node works, and how to use it to receive email messages.

- "Receiving emails" on page 947
- "Receiving an email" on page 948
- "Processing responses from an EmailInput node" on page 950
- "Changing connection information for the EmailInput node" on page 951

Sending emails

You can configure IBM App Connect Enterprise to send an email, with or without attachments, to a static or dynamic list of recipients.

About this task

You can configure the EmailOutput node to send an email, with or without a single attachment, with a static subject and static text, to a static list of recipients. You can also construct a message flow that can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

You can use the SMTP `Server and Port` property of the EmailOutput node to specify the host name of the SMTP server that the integration server uses to send emails, or to specify an alias that refers to an SMTP policy that is defined on the integration server. If the value of the SMTP `Server and Port` property is set to the defined alias, any values that are set in the policy are used in preference to any statically defined value or override value in the local environment.

The order of preference for SMTP value selection is:

1. The value that is specified in the policy if an alias exists with the name that was supplied in the SMTP `Server and Port` property.
2. The server name that is specified in the local environment.
3. The value of the SMTP `Server and Port` property that is specified on the node.

The topics in this section describe the different ways in which you can use the EmailOutput node to send email messages.

Procedure

- To send an email with a static subject and static text to a static list of recipients, see "Sending an email" on page 939.
- To send an email with an attachment, see "Sending an email with an attachment" on page 940.
- To send an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time, see "Producing dynamic email messages" on page 940.
- To send an email that is constructed from a MIME message, see "Sending a MIME message" on page 944.
- To configure the SMTP server, port number, and security identity for the EmailOutput node as an integration node external resource, see "Changing connection information for the EmailOutput node" on page 946.

Sending an email
You can send an email with a static subject and static text to a static list of recipients.

About this task

Use the IBM App Connect Enterprise Toolkit to configure the properties on the EmailOutput node so that you can send an email with a statically defined subject and text, and no attachments, to a statically defined list of recipients. The same email is sent to the same recipients. This method is useful when you want to test the EmailOutput node, or when notification alone is sufficient.

Procedure

1. Switch to the Integration Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.
   a) On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
   b) On the **Email** tab, add the email addresses of recipients by using the `To Addresses`, `Cc Addresses`, and `Bcc Addresses` properties.
   c) On the **Email** tab, add the email address of the sender by using the `From Address` and `Reply-To Address` properties.
   d) On the **Email** tab, provide a subject for the email by using the `Subject of email` property.
   e) On the **Email** tab, provide the text of the email by using `Email message text` property.
4. Save the changes.
5. Add the message flow to the BAR file and deploy.

Results

When a message is passed into the deployed EmailOutput node, an email is sent to the defined set of recipients, containing the subject and text specified on the node.

Sending an email with an attachment
You can send an email with a fixed subject and fixed text, and an attachment, to a static list of recipients.

About this task

You can configure an EmailOutput node to send an email with a single attachment. To send an email with multiple attachments, see "Producing dynamic email messages" on page 940.

Use the IBM App Connect Enterprise Toolkit to configure the properties on the EmailOutput node so that you can send an email with a statically defined subject and text, and an attachment, to a statically defined list of recipients. This method causes the email message to be constructed as a MIME message. The subject, text, and list of recipients remains static, but the content of the attachment is sought dynamically from the message that is passed to the EmailOutput node at run time. The location of the attachment in the message is defined statically.

Procedure

1. Switch to the Integration Development perspective.
2. Add an EmailOutput node to your message flow.
3. Edit the following properties of the EmailOutput node.
   a) On the **Basic** tab, add the SMTP Server and Port information (for example, *smtp.server.com:25*).
   b) On the **Email** tab, add the email addresses of recipients by using the `To Addresses`, `Cc Addresses`, and `Bcc Addresses` properties.
   c) On the **Email** tab, add the email address of the sender by using the `From Address` and `Reply-To Address` properties.
   d) On the **Email** tab, provide a subject for the email by using the `Subject of email` property.
   e) On the **Email** tab, provide the text of the email by using `Email message text` property.
   f) On the **Attachment** tab, set the `Attachment Content` property by using either an ESQL or XPath expression, referring to an element in the message tree; for example, *Body.BLOB*.
   g) On the **Attachment** tab, set the `Attachment Content Name` property with the name of the attachment as it appears in the email.
   h) On the **Attachment** tab, set the `Attachment Content Type`, `Attachment Content Encoding`, and `Multipart Content Type` properties to determine the type of attachment that is sent in the MIME message.
4. Save the changes.
5. Add the message flow to the BAR file and deploy.

Results

When a message is passed into the deployed EmailOutput node, an email is sent to the defined set of recipients, containing the subject, text, and attachment specified on the node.

Producing dynamic email messages
You can produce an email where the SMTP server, list of recipients, subject, text, and multiple attachments are all determined at run time.

About this task

You can create a message flow that produces an email with multiple attachments. To configure an EmailOutput node to send an email with a single attachment, static subject, and static text to a static list of recipients, see "Sending an email with an attachment" on page 940.

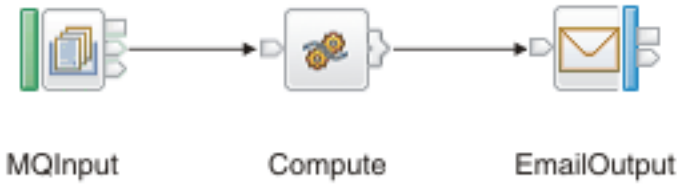
The node properties that you set when sending an email can be optional, and can be overridden at run time by values that you specify in the local environment, email output header (EmailOutputHeader), or the body of the message. To use this method, previous nodes in the message flow must construct these

overrides. Where a text value is not specified in the node properties for the main body of the email, the body of the message that is passed to the EmailOutput node is used.

The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node

About this task




```
CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Add recipient information to the EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.To = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Cc = '<recipient email address>';
  SET OutputRoot.EmailOutputHeader.Bcc = '<recipient email address>';

  -- Add sender information to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.From = '<sender email address>';
  SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply email address>';

  -- Add subject to EmailOutputHeader
  SET OutputRoot.EmailOutputHeader.Subject = 'Replaced by ESQL compute node.';

  -- Add SMTP server information to the LocalEnvironment
  SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

  -- Create a new message body, which will be sent as the main text of the email.
  SET OutputRoot.BLOB.BLOB = CAST('This is the new text for the body of the email.' AS BLOB CCSID 1208);


  RETURN TRUE;
 END;
```

Note: When using a BLOB parser on z/OS, the CCSID of the message should be 500. In the previous example, the CAST would be:

```
SET OutputRoot.BLOB.BLOB = CAST('This is the new text for the body of the email.' AS BLOB CCSID
 500);
```

To write white space characters (such as newline (NL), carriage return (CR), and line feed (LF) characters) in the text string that is produced as the email body, you can add the following lines of code.

```
DECLARE crlf CHAR CAST(X'0D0A' AS CHAR CCSID 1208);
DECLARE myEmailBodyTxt CHAR;
SET myEmailBodyTxt [equals char] 'this is the first line' || crlf ||
    'this is the second line' || crlf ||
    'this is the third line';
SET OutputRoot.BLOB.BLOB = CAST(myEmailBodyTxt AS BLOB CCSID 1208);
```

Using a JavaCompute node

About this task


```
public void evaluate(MbMessageAssembly assembly) throws MbException {
  MbOutputTerminal out = getOutputTerminal("out");

  // Create a new assembly to propagate out of this node, as we want to update it
  MbMessage outMessage = new MbMessage();
  copyMessageHeaders(assembly.getMessage(), outMessage);
  MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
  MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
  MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList,
 outMessage);
  MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

  // Create the EmailOutputHeader parser. This is where we add recipient, sender and subject information.
  MbElement root = outMessage.getRootElement();
  MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

  // Add recipient information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient email address");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient email address");

  // Add sender information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender email address>");
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply email address>");


  // Add subject information to EmailOutputHeader
  SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Replaced by Java compute
 node.");

  // Create Destination.Email. This is where we add SMTP server information
  MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
  MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);

  destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer",
 "<smtp.server:port>");

  // Set last child of root (message body)
  MbElement BLOB = root.createElementAsLastChild(MbBLOB.PARSER_NAME);
  String text = "This is the new text for the body of the email";
  BLOB.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes());

  outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);

  out.propagate(outAssembly);  }
```

Using the local environment

About this task

Use the local environment to specify overrides to the SMTP server connection information and attachments.

Local environment	Description
Destination.Email.SMTPServer	The Server:Port of the SMTP server. Port is optional; if you do not specify it, the default value is 25.

Local environment	Description
Destination.Email.SecurityIdentity	The security identity for authentication with the SMTP server, which can be the name of the userid and password pair that is defined using the **mqsisetdbparms** command, or it can reference an external resource that has a securityIdentity attribute that references a userid and password that are defined using the **mqsisetdbparms** command. In both cases, the value is appended after the string "smtp::". For example, if you use the **mqsisetdbparms** command to create a userid and password of *smtp::myUseridPassword*, the securityIdentity that is specified on the node, or indirectly in an external resource, is *myUseridPassword*.
Destination.Email.BodyContentType	Identifies that the body of the email message contains HTML rather than plain text. You can set this property to text/plain, text/html, or text/xml; text/plain is the default value.
Destination.Email.MultiPartContentType	The type of multipart, including related, mixed, and alternative. You can set any value here.
Destination.Email.Attachment.Content	Either the actual attachment (BLOB/text), or an XPath or ESQL expression that references an element; for example, an element in the message tree or LocalEnvironment. The value of the referenced element is taken as the content of the attachment. • If the element is a BLOB, it is an attachment. • If the element is text, check to see if it can be resolved to another element in the message tree or LocalEnvironment. If it can be resolved, use that element. If it cannot be resolved, add this element as the attachment.
Destination.Email.Attachment.ContentType	The type of attachment (also known as Internet Media Type), including text/plain, text/html, and text/xml. You can set any value here.
Destination.Email.Attachment.ContentName	The name of the attachment.
Destination.Email.Attachment.ContentEncoding	The encoding of the attachment: 7bit, base64, or quoted-printable. • 7bit is the default value that is used for ASCII text. • Base64 is used for non ASCII, whether non English or binary data. This format can be difficult to read. • Quoted-printable is an alternative to Base64, and is appropriate when most of the data is ASCII with some non-ASCII parts. This format is more readable; it provides a more compact encoding because the ASCII parts are not encoded.

Using the email output header

About this task

Use the email output header to specify overrides to the SMTP server connection information and attachments. The EmailOutputHeader is a child of Root. Values that you specify in this header override equivalent properties that you set on the EmailOutput node. Use the SMTP output header to specify any of the email attributes, such as its recipients.

Location	Description
Root.EmailOutputHeader.To	A comma-separated list of email addresses.
Root.EmailOutputHeader.Cc	A comma-separated list of email addresses.
Root.EmailOutputHeader.Bcc	A comma-separated list of email addresses.
Root.EmailOutputHeader.From	A comma-separated list of email addresses.
Root.EmailOutputHeader.Reply-To	A comma-separated list of email addresses.
Root.EmailOutputHeader.Subject	The subject of the email.
Root.EmailOutputHeader.Sender	A single email address. This header is mandatory if multiple email addresses are specified in the Root.EmailOutputHeader.From field.

Any other children of the Root.EmailOutputHeader element are treated as custom headers and are also added to the outgoing email.

Sending a MIME message
You can send an email that is constructed from a MIME message.

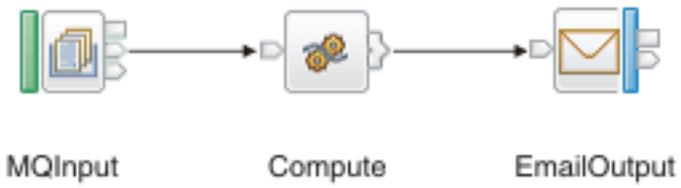
About this task

You can pass a MIME message to the EmailOutput node, which uses the MIME parser to write the MIME message to a bit stream. This message is then sent to the list of recipients in the EmailOutputHeader. Local environment overrides are not considered when a MIME message is passed. You do not need to configure the EmailOutput node in the following examples.

The following examples show how to set up the recipient, sender, subject, SMTP server, and message body information in ESQL (with a Compute node) and Java (with a JavaCompute node).

Using a Compute node

About this task




```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

   CALL CopyMessageHeaders();

   -- Add recipient information to the EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.To = '<recipient email address>';
   SET OutputRoot.EmailOutputHeader.Cc = '<recipient email address>';
   SET OutputRoot.EmailOutputHeader.Bcc = '<recipient email address>';

   -- Add sender information to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.From = '<sender email address>';
   SET OutputRoot.EmailOutputHeader."Reply-To" = '<reply email address>';

   -- Add subject to EmailOutputHeader
   SET OutputRoot.EmailOutputHeader.Subject = 'Dynamic MIME message in ESQL.';

   -- Add SMTP server information to the LocalEnvironment
   SET OutputLocalEnvironment.Destination.Email.SMTPServer ='<smtp.server:port>';

   -- Create a new MIME message body, which will be sent as the main text of the email,
```

```
   -- including an attachment.
   CREATE FIELD OutputRoot.MIME TYPE Name;
   DECLARE M REFERENCE TO OutputRoot.MIME;

   -- Create the Content-Type child of MIME explicitly to ensure the correct order. If we set
   -- the ContentType property instead, the field could appear as the last child of MIME.
   CREATE FIELD M."Content-Type" TYPE NameValue VALUE 'multipart/related; boundary=myBoundary';
   CREATE FIELD M."Content-ID"   TYPE NameValue VALUE 'new MIME document';

   CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P1 REFERENCE TO M.Parts.Part;

   -- First part:
   --   Create the body of the email.
   --   The body of the email has the text 'This is the main body of the email.'.
   CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii';
   CREATE FIELD P1."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
   CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(CAST('This is the main body of the email.'
AS BLOB CCSID 1208));

   CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
   DECLARE P2 REFERENCE TO M.Parts.Part[2];

   -- Second part:
   --   Create the attachment of an email.
   --   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
   CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain; charset=us-ascii;
 name=attachment.txt';
   CREATE FIELD P2."Content-Transfer-Encoding" TYPE NameValue VALUE '8bit';
   CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
   CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(CAST('This is an attachment.' AS BLOB CCSID 1208));

   RETURN TRUE;
END;
```

Using a JavaCompute node

About this task


```
public void evaluate(MbMessageAssembly assembly) throws MbException {
   MbOutputTerminal out = getOutputTerminal("out");
   MbOutputTerminal fail = getOutputTerminal("fail");

   // Create a new assembly to propagate out of this node, as we want to
   // update it
   MbMessage outMessage = new MbMessage();
   copyMessageHeaders(assembly.getMessage(), outMessage);
   MbMessage outLocalEnv = new MbMessage(assembly.getLocalEnvironment());
   MbMessage outExceptionList = new MbMessage(assembly.getExceptionList());
   MbMessageAssembly outAssembly = new MbMessageAssembly(assembly, outLocalEnv, outExceptionList,
 outMessage);
   MbElement localEnv = outAssembly.getLocalEnvironment().getRootElement();

   // Create the EmailOutputHeader parser. This is where we add recipient,
   // sender and subject information.
   MbElement root = outMessage.getRootElement();
   MbElement SMTPOutput = root.createElementAsLastChild("EmailOutputHeader");

   // Add recipient information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "To", "<recipient email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Cc", "<recipient email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Bcc", "<recipient email address>");

   // Add sender information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "From", "<sender email address>");
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Reply-To", "<reply email address>");
```

```
   // Add subject information to EmailOutputHeader
   SMTPOutput.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Subject", "Dynamic MIME message in
 Java.");

   // Create Destination.Email. This is where we add SMTP server information.
   MbElement Destination = localEnv.createElementAsLastChild(MbElement.TYPE_NAME, "Destination", null);
   MbElement destinationEmail = Destination.createElementAsLastChild(MbElement.TYPE_NAME, "Email", null);
   destinationEmail.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "SMTPServer",
 "<smtp.server:port>");

   // Set last child of root (message body) as MIME.
   MbElement MIME = root.createElementAsLastChild("MIME");

   // Create the Content-Type child of MIME explicitly to ensure the correct order.
   MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "multipart/related;
boundary=myBoundary");
   MIME.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-ID", "new MIME document");
   MbElement parts = MIME.createElementAsLastChild(MbElement.TYPE_NAME, "Parts", null);
   MbElement part, data, blob;
   String text;

   // First part:
   //   Create the body of the email.
   //   The body of the email has the text 'This is the main body of the email.'.
   part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-
ascii");
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
   data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
   blob = data.createElementAsLastChild("BLOB");
   text = "This is the main body of the email.";
   try {
  blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
   } catch (UnsupportedEncodingException e) {
  fail.propagate(outAssembly);
   }

   // Second part:
   //   Create the attachment of an email.
   //   The attachment is called 'attachment.txt' and contains the text 'This is an attachment.'.
   part = parts.createElementAsLastChild(MbElement.TYPE_NAME, "Part", null);
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Type", "text/plain; charset=us-
ascii;
name=attachment.txt");
   part.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Content-Transfer-Encoding", "8bit");
   data = part.createElementAsLastChild(MbElement.TYPE_NAME, "Data", null);
   blob = data.createElementAsLastChild("BLOB");
   text = "This is an attachment.";
   try {
  blob.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB", text.getBytes("UTF8"));
   } catch (UnsupportedEncodingException e) {
  fail.propagate(outAssembly);
   }

   outMessage.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
   out.propagate(outAssembly);
}
```

Changing connection information for the EmailOutput node

You can configure the SMTP server, port number, and security identity for the EmailOutput node as an integration node external resource.

About this task

Use an alias that is specified in the SMTP `Server and Port` property on the EmailOutput node. The security identity refers to a user ID and password pair that is defined on the integration server by using the **mqsisetdbparms** command. As documented for the **mqsisetdbparms** command, you must stop and start each integration server that uses a particular DSN, before the information that was modified by the **mqsisetdbparms** command is read and used by that integration server.

Procedure

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create an SMTP policy for the alias that is specified on the node (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Set the SMTP `server name` property to an appropriate value in the form *server:port*.

   The port value is optional; if you do not specify it, the default value is 25. You can also use the SMTP policy to specify a security identity that can be resolved at run time to a user ID and password for authentication with the SMTP server. If you specify a policy name, use the format *{policyProjectName}:policyName*. You can use the **mqsisetdbparms** command to define the security identity at run time.

Receiving emails

You can configure the EmailInput node to receive an email, with or without an attachment, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

About this task

You can use the `Email server` property on the EmailInput node to specify the protocol, host name, and port of the email server that the integration server uses to receive emails. Alternatively, you can specify a policy name that refers to an Email Server policy that is defined on the integration server. If the value of the `Email server` EmailInput node property is set to the defined policy name, any values set by the administrator on the command line are used in preference to any statically defined value.

The following list details the order of preference for value selection:

1. The email server URL value that is specified in the Email Server policy `Email server name` property, if a policy exists that matches the name that is supplied in the `Email server` EmailInput node property.
2. The email server URL value of the `Email server` property that is specified directly on the EmailInput node.

While most policy properties are set by using the Policy editor, security identity support, such as an email server user ID and password pair, is typically set by using the **mqsisetdbparms** command. Security identity support can be configured by setting the `Security identity` EmailInput node property or Email Server policy `Security identity` property to reference a security identity object. However, the security identity object must first be created by using the **mqsisetdbparms** command, for example:

```
mqsisetdbparms INODE -n email::mySecurityIdentity -u myUserID -p myPassword
mqsichangeproperties INODE -c EmailServer -o myEmailConfigurableServiceName -n
securityIdentity -v mySecurityIdentity
```

For more information about email server security identity support, see command.

The topics in this section describe the different ways in which you can use the EmailInput node to receive email messages.

Procedure

- To receive an email, see "Receiving an email" on page 948.
- To configure the email server URL and security identity for the EmailInput node as an integration node external resource, see "Changing connection information for the EmailInput node" on page 951.

Receiving an email
You can receive an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Before you begin

This topic assumes that you have already created a message flow. For more information, see "Creating a message flow" on page 574.

About this task

Use the IBM App Connect Enterprise Toolkit to configure the properties on the EmailInput node so that you can receive an email, with or without attachments.

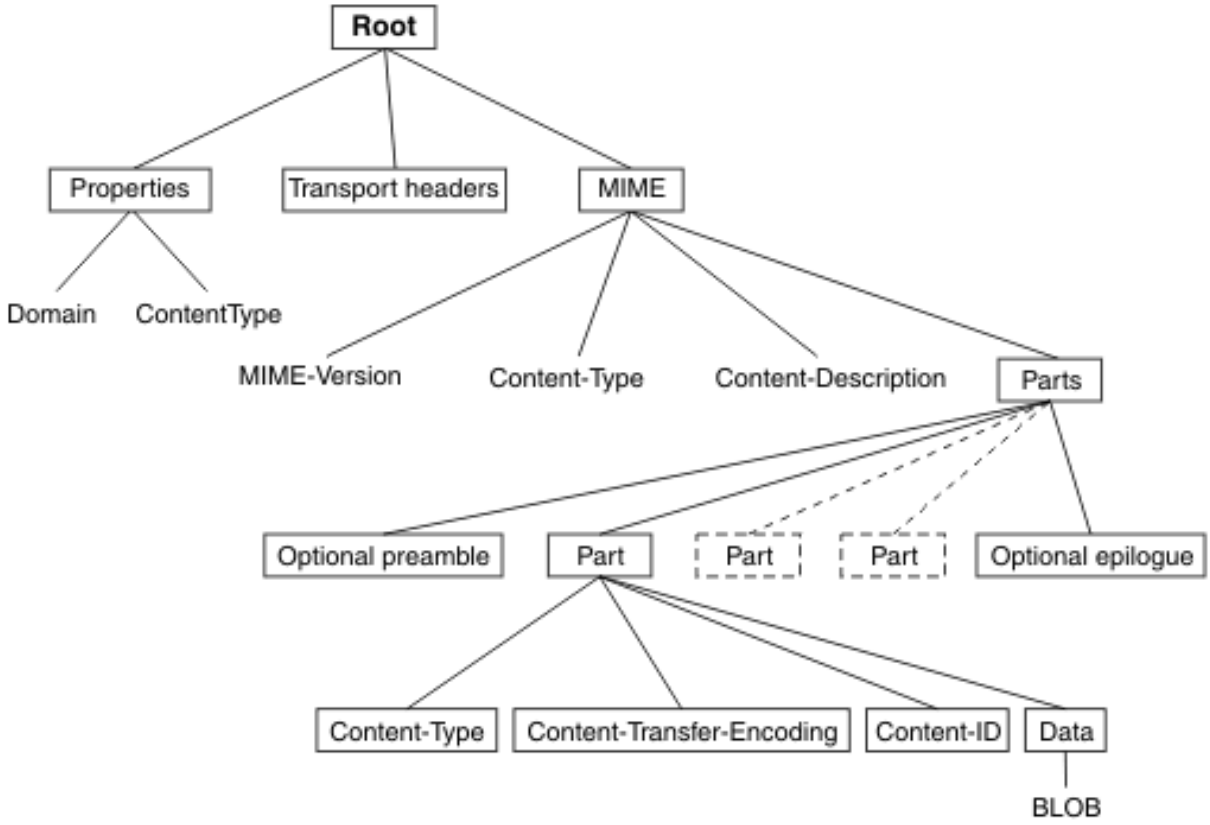
Procedure

1. Add an EmailInput node to your message flow.
2. Edit the following EmailInput node properties:
   a) On the **Basic** tab, add the email server URL, or the Email Server policy name as the `Email server` property value, as described in node. For example, `pop3://myemailserver.com:12345` or `imap://myemailserver.com:56789`.
   b) On the **Security** tab, add the security identity object name of the email server user ID and password pair as the `Security identity` property value. For more information about email server security identity support, see command.
   c) Configure the following properties on the **Retry** tab:
      - `Retry mechanism`: The Retry mechanism property defines how the EmailInput node handles a message flow failure. Valid values are `Failure`, `Short Retry`, or `Short and Long Retry`. The default value for this property is `Short and long retry`, which indicates that the email is retried until the short retry threshold is met, then long retry takes place, meaning that the email is never deleted from the email server, but also that the email is infinitely retried. Emails are deleted from the email server if the email message fails and this property value is not set to `Short and Long Retry`.
      - `Retry threshold`: The Retry threshold property is the number of times to try the message flow transaction again when the `Retry mechanism` property value is set to `Short Retry`. The default value for this property is `0`.
      - `Short retry interval (in seconds)`: The Short retry interval is the interval, in seconds, between each retry if the `Retry threshold` property value is not set to zero. The default value for this property is `0`. If the email is retried until the short retry threshold is met and the email fails, the email is routed to the failure terminal, and the email is deleted from the email server.
      - `Long retry interval (in seconds)`: The Long retry interval is the interval, in seconds, between each retry, if the `Retry mechanism` property value is `Short and Long Retry`, and the short retry threshold has been exhausted. The default value for this property is `300` seconds.
      - `Action on failing email`: The Action on failing email property determines the action that the EmailInput node takes with the input data source after all attempts to process the email contents fail. The Action on failing email property is a read only property that is set to a default value of `Delete Email`, which is used with the Retry mechanism property. If the Retry mechanism property is set to `Short and Long Retry`, the message flow continues to try to retrieve the email from the email server, meaning that the email is never deleted. If the Retry mechanism property is not set to `Short and Long Retry`, the Action on failing email property value `Delete Email` is used, and the email is deleted from the email server.
3. Save the changes.
4. Add the message flow to the BAR file and deploy.

   When a message is passed into the deployed EmailInput node, an email is received from the email server and the body of the email message, and any attachments, are propagated in the Multipurpose

Internet Mail Extensions (MIME) domain. All other information relating to the email is stored in the `Root.EmailInputHeader` MIME logical tree. For a complete list of the email elements that are propagated in the MIME logical tree when you use an EmailInput node, see node.

When an email containing an attachment is received, the EmailInput node places different parts of the email body in the MIME domain, so that they are associated with the MIME parser. The MIME tree location that the EmailInput node builds to house the information is the same location that the EmailOutput node expects email data to be in when sending an email. The attachment is stored in the MIME logical tree in directory `Root.MIME.Parts.Part.Data`. Where Content-Type describes the type of data that is in the attachment.



Viewing the `Root.Properties.ContentType` value in the MIME domain allows you to write logic to parse the attachment. For more information about the MIME logical tree, see "MIME parser and domain" on page 547.

Results

Received emails are deleted from an email server that supports POP3 or IMAP only when the emails have been successfully propagated after being processed by the EmailInput node Failure, Out, or Catch terminals, and the message flow has successfully executed. This does not form part of a globally coordinated transaction.

Emails are deleted from the email server under the following circumstances:

- The Failure terminal is not connected.
- An exception occurs in the Failure terminal.
- The email message fails and the `Retry mechanism` property value is not set to `Short and Long Retry`.
- The `Retry threshold` is not set to `0` and the `Short retry interval` property value has been exhausted.

For more information about processing responses from an EmailInput node, and for information about rollback handling, see "Processing responses from an EmailInput node" on page 950.

Processing responses from an EmailInput node
The EmailInput node can return different response messages that indicate the success or failure of receiving an email, with or without attachments, from an email server that supports Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP).

Before you begin

Ensure that you have developed a message flow with an EmailInput node, as described in "Receiving an email" on page 948.

About this task

The EmailInput node has three output terminals:

- Failure: The output terminal to which the message is routed if an EmailInput node failure is detected when a message is propagated, or an EmailInput node fails to access the email server. Connect the Failure terminal of this node to another node in the message flow to process errors.
- Out: The output terminal to which the message is routed if it has been propagated successfully. Connect the Out terminal of this node to another node in the message flow to process the message further, or send the message to an additional destination.
- Catch: The output terminal to which a message is routed if an exception is thrown downstream and caught by this node. Exceptions are caught only if this terminal is attached.

Procedure

- **Processing successful returns**

  When an EmailInput node successfully receives an email, the resulting message is propagated to the Out terminal.

- **Processing downstream message flow exceptions**

  If a message flow exception occurs downstream of the Failure terminal in the message flow, a message is routed to the Catch terminal. If you do not have the `Retry mechanism` property value set to `Short and Long Retry` on the EmailInput node Retry tab, the current transaction is rolled back.

- **Handling failures in the node**

  Any other failures are propagated to the Failure terminal. Possible failures include:

  – A problem retrieving an email message. For example, an inability to communicate with the target email server because of a connection or authentication error. If this occurs, the connection is re attempted when the polling interval expires. Each connection failure occurrence causes an appropriate message in user trace.
  – A problem parsing an email.
  – An internal EmailInput node exception or error is detected before the message is propagated to the Out terminal. For example, a malformed email is received, causing the EmailInput to propagate the message and an exception list to the Failure terminal.
  – The `Retry mechanism` property value is set to `Failure`, causing an immediate message to be sent to the Failure terminal.
  – The `Retry threshold` is not set to `0` and the `Short retry interval` property value has been exhausted, causing a message to be routed to the failure terminal, and the email being deleted from the email server.

  If the failure terminal is not connected, or an exception occurs down the Failure terminal, or the email message fails and you have not set the `Retry mechanism` property value to `Short and Long Retry`, the email is deleted from the email server.

Results

You can use the Email Server policy to change connection details for the EmailInput node (see "Changing connection information for the EmailInput node" on page 951).

Changing connection information for the EmailInput node

You can create a policy that the EmailInput node or message flow refers to at run time for email server connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and security identity values without needing to redeploy your message flow.

Before you begin

- Read "Receiving emails" on page 947 for background information.

About this task

Use the Email Server policy to change the email server connection information for the EmailInput node.

Procedure

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **Email Server** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Email server` property on the EmailInput node.
3. Set the `Email server name` property in the policy to the URL of the email server that contains incoming email messages.

   The URL must be in the format `protocol://hostname:port`, where:

   - `protocol` can be `pop3`, `pop3s`, `imap`, or `imops`.
   - `hostname` is the Internet Protocol version 4 (IPv4) TCP/IP address or DNS-resolvable host name of the email host.
   - `port` is the port number that the email server is listening on for connections over Post Office Protocol 3 (POP3) or Internet Message Access Protocol (IMAP). You can enter an integer in the range `1 - 65535`.

   For example:

   ```
   pop3://myemailserver.com:12345
   imap://myemailserver.com:56789
   ```

   For more information, see Email Server policy (EmailServer).
4. Set the `Security identity (DSN)` property in the policy to the name of the security identity object that contains the user ID and password that is used to authenticate the connection to the email server.
5. Set the timeout properties in the policy file to avoid hangs due to socket wait calls. For example:

   - Set **`Connection timeout (seconds)`** to 5.
   - Set **`Read timeout (seconds)`** to 5.
   - Set **`Connection idle timeout (seconds)`** to 300.
6. Deploy this policy before you start the associated message flow.

Working with files

You can use the FileInput, FileRead, CDInput, and FTEInput nodes in your message flows to process data from files. You can use the FileOutput node, CDOutput, and FTEOutput node to send data from a message flow into a file.

About this task

Using files is one of the most common methods of storing data. You can create message flows to process data in files, accepting data in files as input message data, and producing output message data for file-based destinations. The following file nodes are provided:

- FileInput node. Use this node to receive messages from files in the file system of the integration node or, by using FTP or SFTP, in a remote file system. The node generates output message data that any of the output nodes can use, which means that messages can be generated for clients using any of the supported transport protocols to connect to the integration node. For more information, see "Using a local file as input for your message flow" on page 975.
- FTEInput node. Use this node to start a message flow when files are received over a IBM MQ File Transfer Edition network. For more information, see "Receiving a file by IBM MQ File Transfer Edition" on page 983.
- CDInput node. Use this node to start a message flow when files are received over a IBM Sterling Connect:Direct network. For more information, see "Receiving a file using IBM Sterling Connect:Direct" on page 984.
- FileRead node. Use this node to read data from a file in the middle of a message flow. For more information, see "Routing or enriching a message based on the contents of a file" on page 976.
- FileOutput node. Use this node to write messages to a file in the file system of the integration node or, by using FTP or SFTP, in a remote file system. The node can create new files, replace existing files and append data to the end of an existing file. For more information, see "Writing a file to your local file system" on page 989 and "Writing a file to a remote FTP or SFTP server" on page 991.
- FTEOutput node. Use this node to send a file to a remote destination by using a IBM MQ File Transfer Edition network. For more information, see "Sending a file by IBM MQ File Transfer Edition" on page 994.
- CDOutput node. Use this node to send a file to a remote destination by using an IBM Sterling Connect:Direct network. For more information, see "Initiating a managed file transfer using IBM Sterling Connect:Direct" on page 1003.

The FileInput and FTEInput nodes start the message flow when a new file arrives, whereas the FileRead node must be connected to another node to start the message flow transaction. The FileRead node also provides keyed access to identify a record, unlike the FileInput node, which processes all records in order.

By using these five nodes, you can also process large files without the complete message being held in memory, and you can simplify the processing of files that have large numbers of repeating entries.

If you are accessing files on an NFS share, ensure that you are using NFS version 4. The server must support file locking.

If you want to work with files, read these topics:

- "How multiple file nodes share access to files in the same directory" on page 960
- "Using local environment variables with file nodes" on page 962
- "File name patterns" on page 972
- "Archiving" on page 974
- "Reading files" on page 974
- "Writing a file" on page 989
- "Transferring files securely by using SFTP" on page 998
- "Managed file transfers using WebSphere MQ File Transfer Edition" on page 1000
- node
- node
- node
- node
- node

- node
- node

IBM Sterling Connect:Direct overview and concepts

An overview of IBM Sterling Connect:Direct and its terminology when used with IBM App Connect Enterprise.

IBM Sterling Connect:Direct is a managed file transfer product that transfers files between, and within, enterprises.

IBM Sterling Connect:Direct, in conjunction with IBM App Connect Enterprise uses the following terminology:

Connect:Direct server

An application that runs on a machine requiring IBM Sterling Connect:Direct functionality. Files are transferred between two Connect:Direct servers.

- The primary Connect:Direct server, also referred to as the PNODE.
- The secondary Connect:Direct server, also referred to as the SNODE, that receives the transferred file and places the file on the local file system.

Connect:Direct-S

Connect:Direct server which transfers files to other Connect:Direct servers.

Connect:Direct-R

Connect:Direct application that connects to a Connect:Direct server and requests that server does some form of processing. IBM App Connect Enterprise acts as a Connect:Direct-R to request transfers or receive information about transfers.

IBM App Connect Enterprise CDInput node

Use the IBM App Connect Enterprise CDInput node to receive messages that have been transferred to a given Connect:Direct server.

If two or more input nodes are deployed to listen on the same Connect:Direct server, only one receives the file and which one that is cannot be determined. There is some distribution between the nodes, and this is equivalent to having two or more MQInput nodes listening on the same queue.

CDInput nodes can be used in different flows, and in different integration servers, against the same Connect:Direct server.

You can configure the CDInput node to process only a subset of files transferred to the server, based on the directory name and the file name. This allows for multiple CDInput nodes within the same integration server to receive specific files, depending on the filters used.

See “IBM Sterling Connect:Direct concepts” on page 954 for further information on the CDInput node, CDOutput node, and Connect:Direct server.

IBM App Connect Enterprise CDOutput node

Use the IBM App Connect Enterprise CDOutput node to serialize the message tree to a file and then transfer it to a secondary Connect:Direct server using a primary Connect:Direct server. A directory under the work path within the integration server is used as the staging area, until the file is ready to be transferred. Once the file is transferred, it is deleted from the staging area.

Initparm

The definition of the startup parameters for the primary Connect:Direct server (PNODE) and secondary Connect:Direct server (SNODE).

Netmap

The definition of a connection between the primary and secondary Connect:Direct servers.

Processname

The name given to a process that is run in IBM Sterling Connect:Direct. More than one process can be given the same name, and each process is identified uniquely using the process number.

Note that IBM App Connect Enterprise nodes work purely as clients connecting to the external Connect:Direct server, using the IBM Sterling Connect:Direct Java Application Interface.

By default, IBM App Connect Enterprise attempts to connect to a local Connect:Direct server, on the default port:

- When sending a file using CDOutput nodes, or
- To be notified of files to process using CDInput nodes.

To enable the CDOutput and CDInput nodes to connect to the local Connect:Direct server, you must set up a username and password using the command.

IBM Sterling Connect:Direct concepts

The CDInput node receives messages that have been transferred to a given Connect:Direct server.

The node receives both the contents of the file and meta data provided by IBM Sterling Connect:Direct on the transfer. One or more CDInput nodes can be used to receive transfers, either in the same flow, different flows, or different integration servers; for any given transfer only one CDInput node receives a message.

You can also specify which transfer a CDInput node can receive, using filters based on directory and file name of the transfer. Once the transfer has been processed there are a set of options of what to do with the transferred file; for further details see node.

A simple flow that takes all transferred files from a Connect:Direct server and writes them to an IBM MQ queue could contain a CDInput node and an MQOutput node.

Once the CDInput node has been notified of a file to be processed, it processes the file in the same way that a normal FileInput node does. The CDInput node is notified by the Connect:Direct server manager when a transfer occurs. The notification message is persistently stored on IBM MQ and survives queue manager restarts.

The Connect:Direct server manager is defined and runs using a Connect:Direct Server policy. This policy is used for administering the access, and processing files using IBM Sterling Connect:Direct (see Connect:Direct Server policy (CDServer)).

Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the CDInput and CDOutput nodes.

You must also create the system queues that are required by the CDInput and CDOutput nodes, including the two queues that are used to store transfer details from the Connect:Direct server:

- SYSTEM.BROKER.CD.STATS
- SYSTEM.BROKER.CD.TRANSFERS

For information about creating the system queues, see “Creating the default system queues on an IBM MQ queue manager” on page 99.

For each successful transfer to the Connect:Direct server, a message is placed on the TRANSFERS queue, which is used to trigger the CDInput node to process the file. The STATS queue contains internal data used by IBM App Connect Enterprise to record which transfers have been processed by IBM App Connect Enterprise. Clearing both sets of these queues causes all transfers that have occurred up to the current time to be ignored; there is no need, for example, for a restart.

The input node does not poll for transfers, but is triggered instead when they arrive. However, the Connect:Direct server manager does need to poll for events to notify the CDInput node about a transfer. The Connect:Direct server manager has a polling interval of one second.

The CDInput and CDOutput nodes go through the Connect:Direct server manager to both send and receive transfers from a given server, the name of which is set on the node. The Connect:Direct server manager properties are defined by using a Connect:Direct Server policy.

The Connect:Direct server manager organizes:

- Monitoring the Connect:Direct server for transfers which need to be processed by the CDInput nodes.
- Sending commands created by the CDOutput node to the Connect:Direct server to perform transfers.

In both cases, the Connect:Direct server manager can be used to configure the directory structure to a shared file system, used to both send and receive files from the Connect:Direct server.

The high-level structure of the directory might be different on the two different machines and you can define properties on the Connect:Direct server manager to give the correct mapping.

The *Default* policy connects to the Connect:Direct server on the local host by using the default port. You can modify the default policy to a more complex configuration with the CD server on different ports, or a different machine, or change the node to use a completely different policy that is set up to use a different port and host name.

As well as defining connections details, you can also configure other key properties, such as the location of staging directories and mapping between directories on different machines.

The Connect:Direct server manager is started and stopped by the CDInput and CDOutput nodes as required and is not an artifact that is deployed or seen in the deployment view; all of its function is defined by the properties in the policy. If transfers happen while IBM App Connect Enterprise is stopped (or IBM Sterling Connect:Direct flows are stopped) they are not missed and are processed as soon as the flows restart. All transfers are persistent and survive queue manager restarts.

The policy security identity must have adequate permission for the Connect:Direct server manager within the integration server to detect all transfers that are sent to the Connect:Direct server. For example: On UNIX systems, the user's stanza within the `userfile.cfg` in Connect:Direct must have `:cmd.selstats=A:\`. This property can either be set explicitly, or inherited from the higher level property `:admin.auth=y:\`. If this permission is not present, then the CDInput nodes do not detect or process files that were transferred by other users. For more information about the security identifier, see the `Security identity` property in Connect:Direct Server policy (CDServer).

IBM App Connect Enterprise makes a number of API connections to the local Connect:Direct server to process inbound transfers, and to request outbound transfers.

IBM App Connect Enterprise creates one API connection per policy per integration server. This connection enables CDInput nodes to process files sent to the local Connect:Direct server. If there are multiple policies that relate to the same Connect:Direct server, then multiple API connections are established.

In addition to these connections, each message flow instance that uses a CDOutput node also creates an API connection to submit transfer requests to the Connect:Direct server. Where there are multiple flows that contain CDOutput nodes, or where these flows have extra defined instances, IBM App Connect Enterprise creates multiple API connections. If the number of API connections that are required exceeds the maximum that is configured in the Connect:Direct server, then the connection fails causing a BIP7951E exception, containing the following information: "`Error: Logon failed! Attempted to exceed maximum API connections`". Avoid this error by increasing the maximum client connections for the Connect:Direct server. The maximum number of connections that are accepted by a Connect:Direct server on UNIX is controlled by the **`api.max.connects`** parameter. For more information about this parameter, and the equivalent settings on other operating systems, see the IBM Sterling Connect:Direct product documentation: http://www.ibm.com/support/knowledgecenter/SS4PJT_5.2.0/cd_52_welcome.html.

How the integration node processes files

The integration node reads files with the FileInput, FTEInput, CDInput, and FileRead nodes, and writes files with the FileOutput, CDOutput, and FTEOutput nodes.

IBM App Connect Enterprise can read messages from files and write messages to files in the local file system, or on a network file system that is local to the integration node. The following message flow

nodes provide this capability. Also described are the required access permissions for the directories and files they operate on:

- FileInput node: read and write permissions on the input directory and input files.
- FileOutput node: read and write permissions on the output directory.
- FileRead node: read permissions on the input directory and input files. Note: if a Finish File action is used (to move, rename or delete the file), write permissions on the input directory and input files is also required.

The FileInput, FileOutput, and FileRead nodes inherit the permissions assigned to the integration node's user ID and group. See your operating system documentation for details on how to set permissions and the impact on file access.

You can use the FTEInput and FTEOutput nodes to receive or send files to a destination on a IBM MQ File Transfer Edition network.

You can use the CDInput and CDOutput nodes to receive or send files to a destination on an IBM Sterling Connect:Direct network.

A file, or a record within a file, is analogous to a message in a queue. The directory that contains the file is analogous to a message queue.

How the integration node reads a file at the start of a flow

The FileInput node processes messages that are read from files. The FileInput node searches a specified input directory (in the file system attached to the integration node) for files that match specified criteria. The node can also recursively search the input directory's subdirectories. Files that meet the criteria are processed in age order, that is the oldest files are processed first regardless of where they appear in the directory structure. Optionally, files from a remote FTP or SFTP server can be moved to the local directory whenever the directory is to be scanned. You can find the file that you require by specifying an explicit file name or a file name pattern that includes wildcard characters. If the file is locked, it is ignored during the directory scan. You can also exclude files from being read by specifying a file exclusion pattern. Whether files are either processed or ignored is determined first according to the file name or pattern and second according to the file exclusion pattern.

The FileInput node creates an `mqsitransitin` subdirectory in the input directory. The `mqsitransitin` subdirectory holds and locks the input files while they are being processed. The integration node reads the file and propagates a message, or messages, by using the contents of the file.

If an integration server that processes files in this input directory is removed, check the `mqsitransitin` subdirectory for partially processed or unprocessed files. Move any such files back into the input directory and remove the integration server UUID prefix from the file names, so that they can be processed by a different integration server. For more information about the `mqsitransitin` subdirectory, see “How multiple file nodes share access to files in the same directory” on page 960.

In the FileInput node, you can specify how the records are derived from the file. The contents of a file can be interpreted as:

- A single record (whole file)
- Separate records, each of a fixed length (fixed-length records)
- Separate records each delimited by a specified delimiter (delimited records)
- Separate records that are recognized by a parser that you specify (parser record sequence)

After the file has been successfully processed, it is either deleted from the file system or moved to an archive subdirectory of the specified (local) directory.

When the last record of the file has been processed successfully, if the End of Data terminal of the FileInput node is connected in the message flow, an End of Data message is propagated to the End of Data terminal. The End of Data message consists of an empty BLOB message and a `LocalEnvironment.File` structure, and it allows explicit end-of-flow processing to be performed in another part of the message flow.

The message (or messages) propagated from the file can be used as input to any message flow and output node. You can create a message flow that receives messages from files and generates messages for clients that use any of the supported transports to connect to the integration node.

Whenever a message is propagated from a file, the FileInput node stores information about the file in the LocalEnvironment.File message tree. This information includes the offset of the record of the message in the file that is being processed, and the record number in that file. In addition, when a wildcard is used in a file name pattern, the characters matched in the file name are placed in the WildcardMatch element of the local environment tree.

If a file is backed out, or a file is left in the `mqsitransitin` subdirectory because processing failed, remove the integration server UUID prefix from the file name and move it back into the input directory. Processing is automatically retried, beginning at the first record in the file.

The FTEInput node receives files from the IBM MQ File Transfer Edition network. The integration node reads the file and propagates a message, or messages, by using the contents of the file. After an FTEInput node has processed a file, the file is deleted. For details of information that the FTEInput node provides in the local environment, see “Using local environment variables with file nodes” on page 962.

The CDInput node receives files from the IBM Sterling Connect:Direct network. The integration node reads the file and propagates a message, or messages, by using the contents of the file. After a CDInput node has processed a file, the file is deleted. For details of information that the CDInput node provides in the local environment, see “Using local environment variables with file nodes” on page 962.

The FileRead node can read a whole file, or a single, nominated record. A common use of the FileRead node is to route or enrich messages based on the contents of the file. When developing the message flow you can specify the name and location of the file to be read. You can override these values at run time based on the contents of a message. TheFileRead node reads a file in the middle of a message flow.

The FileRead node complements the existing FileInput and FileOutput nodes.

How the integration node reads a file from the middle of a flow

The FileRead node locks files when they are being read and does not move files to a subdirectory to read them. Files can be read completely or separated into records, in the same way as the FileInput node. After the file is processed by the FileRead node, it can be deleted, archived, or left unchanged.

When a FileRead Node has the `Remote Transfer` option selected, and a local file is not available, one is transferred from a remote server. Only one thread on a single integration server is permitted to transfer a particular file at any time. Any message flow invocations that require access to this same file wait until an in-flight transfer is completed and then use the local copy of the file. The FileRead node creates a lock in the `locks` subdirectory of the local input directory to prevent multiple integration servers from transferring the same file at the same time. This locking only functions correctly if the file system correctly implements Posix locking semantics, in particular NFS file systems must be at least NFS v4.

If the `Delete remote file after successful transfer` property is selected, then after successfully transferring a file it is deleted from the remote server, otherwise it is left in place.

If the FileRead Node has the **`Finish File Action`** property set to `No Action`, then it ensures that the input file referenced by the File Name pattern is unique in the input directory. This behavior means that if more than one file matches in the local directory due to a wildcard match, then an exception is thrown. This behavior is preserved for remote files, so if the **`Finish File Action`** is set to `No Action` and multiple files match the file pattern on the remote server, an exception is thrown. Under these conditions, the broker is unable to determine which of the multiple files present should be processed.

How the integration node writes a file

The FileOutput, CDOutput, and FTEOutput nodes write messages to files in the file system of the integration node. When a message is received on the In terminal of the node, it creates and writes a file as a series of one or more records. One record is written to a file for every message received. The name of the file is specified either by a file name pattern in the node or an explicit file name that is either specified in the node or derived from the message.

You can specify how the records are accumulated in files:

- A single record (whole file). The file that is created consists of one record.
- Concatenated records (unmodified records). The records are not padded to any required length, and they are not separated by a delimiter.
- Uniform length records (fixed-length records). Records that are shorter than the specified length are padded to the required length.
- Separate records (delimited records). Records are either terminated or separated by a specified delimiter.

The message flow informs the output node that there are no more records to write by sending a message to its Finish File terminal. The file is then delivered to the appropriate destination:

- FileOutput node: The file is then moved to the specified output directory. If this transfer fails, you can specify if the file is deleted, moved to the failure directory (`mqsifailure`), renamed before being moved to the failure directory, or no action is taken on the file.

  Optionally, the file can be moved to a directory on a remote FTP or SFTP server. The server is identified by the `Remote server and port` property on the node. Alternatively, you can override the node property by setting a value in the local environment (see Local environment overrides for the remote server on the node).
- FTEOutput node: The file is sent to the destination agent on the IBM MQ File Transfer Edition network.
- CDOutput node: The file is sent to the IBM Sterling Connect:Direct network.

If the output node produces a single record from the file (whole file), the file is moved immediately to the output directory without requiring a message to be propagated to the Finish File terminal. In this case, any message sent to the Finish File terminal of the node has no effect on any file, but the message is still propagated to a message flow attached to the End of Data terminal.

How data is appended to the end of a local file

There are two methods of appending to a local file by using the FileOutput node:

- The first method is to write directly to the output directory. Each record that is written is appended immediately to the specified file. The contents of the file can be read while data is being written to the file. The Finish File action, and subsequently the Action on Finial File failure, are not effective unless FTP options are used. If FTP options are used, the Finish File action transfers the file to the remote machine.
- The second method is to stage the file in the `mqsitransit` directory. The records are collected in the file. If a file exists in the output directory when the Finish File action occurs, the file is moved into the transit directory and the data is appended to the end of this file. The file is then moved back to the output directory and the new data is now visible to an application.

On the FTP tab if `Retain local file` and `Stage in the transit directory` are both selected then `Action if file exists` occurs on the Finish File action.

How the FileOutput node deals with failures

You can define the action that the node takes with the output file if the final processing of a file fails. The following table describes the outcome for each action taken by the node on final file processing failure given the write mode and the type of transfer:

Action on final file processing failure	Write Mode	Remote Transfer	Outcome
No Action	Stage in `mqsitransit` directory	Either	File is left in `mqsitransit` subdirectory unless **`Record Definition`** is `Whole File` then File is not written to output directory
No Action	Write directly to the output file	No	File is not written to output directory
No Action	Write directly to the output file	Yes	File is left in output directory unless there is a failure writing to the local output directory
Delete	Stage in `mqsitransit` directory	Either	File is deleted from `mqsitransit` subdirectory
Delete	Write directly to the output file	No	File is not written to output directory
Delete	Write directly to the output file	Yes	File is deleted from the local output directory
Move To `mqsifailure`, or move to `mqsifailure` and add timestampStage in `mqsitransit` directory	Stage in `mqsitransit` directory	Either	File is moved from `mqsitransit` subdirectory to `mqsifailure` subdirectory
Move To `mqsifailure`, or move to `mqsifailure` and add timestampStage in `mqsitransit` directory	Write directly to the output file	No	File is not written to output directory, and `mqsifailure` subdirectory is not created
Move To `mqsifailure`, or move to `mqsifailure` and add timestampStage in `mqsitransit` directory	Write directly to the output file	Yes	File is moved from the local output directory to `mqsifailure` subdirectory unless there is a failure that prevents the file being written to the local output directory. In this case, the file can not be moved to `mqsifailure` as it did not exist in the local output directory.

How data is appended to the end of a remote file

The FileOutput node builds a file either in the output directory or in the `mqsitransit` directory. On the FTP tab, if `Action if remote file exists` is set to Append, when the Finish File action occurs the file is transferred to the remote machine by using the FTP Append verb. The data is appended to the end of the remote file or, if the file does not exist, the file is created.

Recognizing file records as messages to be parsed
Use the FileInput, FTEInput and FileRead nodes to segment your input file into messages that are to be parsed.

The node segments your input file into messages that are to be parsed by one of the following parsers:

- XMLNSC
- MRM Custom Wire Format (CWF)
- MRM Tagged Delimited String Format (TDS)

The `Message domain` property of the node specifies the parser to use; XMLNSC or MRM. Specify `Parsed Record Sequence` for the `Record detection` property so that the node splits the file into messages to be parsed by either the XMLNSC parser or MRM parser.

The XMLNSC parser

If you select the XMLNSC parser, the end of the root tag marks the end of the message. XML comments, XML processing instructions, and white space that appear after the end of the XML message are discarded. The start of the next XML message is marked either by the next XML root tag or the next XML prolog.

The MRM parser

If you select an MRM parser, ensure that the message model has a defined message boundary and does not rely on the parse being stopped when it reaches the end of the bit stream. If the final element has a *maxOccurs* value of -1, the parser continues to read bytes until the end of the bit stream or until it encounters bytes that cause a parsing exception. In either case, the parser is unable to identify the end of one message and the start of the next. If you use *Data Element Separation* = `Use Data Pattern`, ensure that the pattern recognizes a specified number of bytes. Be aware, therefore, that a pattern of `*` identifies all available characters and so would read an entire input file.

If you use delimited separations with message group indicators and terminators, ensure that the combination of group indicator and terminator does not match a record delimiter. For example, a message might start with a left brace ({) and end with a right brace (}). If there is a delimiter of `}{` within the message, the delimiter matches the boundary between multiple messages; as a result, a delimiter within the current message might be identified as a message boundary. This might cause bytes in a subsequent message to be included in the current message causing parser exceptions or unexpected content in the parse tree.

How multiple file nodes share access to files in the same directory

IBM App Connect Enterprise controls access to files so that only one file node at a time can read or write to a file.

When a message flow uses the FileInput or FileOutput node, additional instances (threads) might be associated with the message flow, or file nodes in other message flows in the same or different integration servers might refer to files in the same directory. IBM App Connect Enterprise controls the way in which multiple processes read from and write to files by moving the files to the `mqsitransitin` directory during processing, and locking them while they are being processed. The `mqsitransitin` directory is a subdirectory of the input directory specified in the FileInput node.

If you are using the FileInput node and choose to include subdirectories, an `mqsitransitin` directory is created as needed in all places where there is a file to be processed.

The integration node locks the files that are being read by the FileInput node or written by the FileOutput node, to prevent other integration servers from reading or changing the files while they are being processed. The integration node unlocks the file:

- When a FileInput node finishes processing the input file.
- When a FileOutput node finishes writing the file and moves it from the transit directory to the output directory.

Note: The FTEInput node does not use a transit directory. Each integration server has its own IBM MQ File Transfer Edition agent, and a node processes only files sent to the agent for which the node is deployed. The integration server ensures that only one node in the integration server processes each file.

Reading a file

When the FileInput node reads a file, it first moves the file into the `mqsitransitin` directory, where it is held during processing. A prefix (containing the integration node name and integration server name) is added to the file name to indicate which integration server is processing the file. While the file is in this directory, no other integration servers can access the file. The integration node maintains a `lock` subdirectory in the `mqsitransitin` directory, to ensure that files in the input directory are accessed by only one integration server at a time.

If multiple message flows or instances within an integration server are reading from the same input directory, only one instance of one message flow is allocated to reading it. Each record in the file is serially processed by this instance. Other instances of the message flow, or other message flows, can simultaneously process other files, the names of which match the pattern specified in the `File name or pattern` property of the node.

While a file is being processed, the file system is used to lock the file. As a result, other programs (including other integration servers) are prevented from reading, writing, or deleting the file while it is being processed by the file nodes.

While a FileInput node is reading a file, the file remains in the `mqsitransitin` directory until it has been fully processed (or until an unrecoverable error occurs). If the file is to be retained, it is held in a subdirectory of the `mqsitransitin` directory.

When the file has been processed, it is moved from the `mqsitransitin` directory back to the input directory. However, if the integration server stops unexpectedly while the file is in the `mqsitransitin` directory, you can manually restore the input file to the input directory by removing the prefix (containing the integration node name and integration server name) from the file name, and then moving it to the input directory. The input file is then processed by the next FileInput node that scans the directory.

If you use an NFS server, and have File nodes in different integration servers that access the same directory on the NFS server, ensure that you are using NFS version 4 to correctly support file locking.

Writing a file

Files that are created and written by a FileOutput node are put in the output directory when they are finished. While records are being added to a file, it is kept in the `mqsitransit` subdirectory.

Each record is written by a single message flow instance. All message flow instances that are configured to write records to a specific file can append records to that file. Because instances can run in any order, records that they write might be interleaved, which means that the sequence of records might be altered. If you require the sequence of records in the output file to be maintained, ensure that only one FileOutput node instance uses the file. To ensure that only one FileOutput node instance uses the file, configure the message flow that contains the node to use the additional instances pool with zero instances, and ensure that other message flows do not write to the same file.

While a file is being processed, the file system is used to lock the file. As a result, other programs (including other integration servers) are prevented from reading, writing, or deleting the file while it is being processed by the file nodes. This lock is retained for a short period after a FileOutput node writes to the file without finishing it, leaving it in the transit directory. If message flows that are in the same integration server use the same output file and run sufficiently quickly, the integration node does not relinquish the lock before the file is finished. However, if the message flows have longer intervals between them, the integration node relinquishes the lock and another process or integration server can acquire a lock on the file. To prevent this situation, ensure that output directories are not shared between integration servers.

Using local environment variables with file nodes

You can use fields in the local environment to dynamically alter the behavior of the FileInput, FileOutput, FTEInput, and FTEOutput nodes. You can also find what values the output nodes used to process the file.

These fields are available in the following message tree structures:

- LocalEnvironment.File
- LocalEnvironment.File.Read
- LocalEnvironment.WrittenDestination.File
- LocalEnvironment.Destination.File
- LocalEnvironment.Destination.File.Remote
- LocalEnvironment.Wildcard.WildcardMatch
- LocalEnvironment.FTE
- LocalEnvironment.WrittenDestination.FTE
- LocalEnvironment.Destination.FTE
- LocalEnvironment.CD
- LocalEnvironment.CD.Transfer
- LocalEnvironment.Destination.CD
- LocalEnvironment.WrittenDestination.CD

LocalEnvironment.File fields

When you use the FileInput node, it stores information that you can access in the LocalEnvironment.File message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the input node started processing the file in the Coordinated Universal Time (UTC) zone, as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
The following elements contain data about the current record:		
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0. If this element is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. If this element is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. If this element is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.

Element Name	Element Data Type	Description
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. It is set to TRUE if the current record is empty. If this element is part of the End of Data message tree, this value is always set to TRUE.

This structure is propagated with each message written to the Out terminal of the FileInput node and with the empty message written to the End of data terminal.

LocalEnvironment.File.Read fields

When the FileRead node propagates a message, it stores valid information about it in the LocalEnvironment.File.Read message tree. If the input file is empty, an empty message is propagated. The following table lists the LocalEnvironment.File.Read message tree structure.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`). Alternatively this path relates to the file nodes root directory, which can be overridden with the same environment variable as used for the FileInput and FileOutput nodes.
Name	CHARACTER	File name and extension.
LastModified	TIMESTAMP	Date and time the file was last modified.
TimeStamp	CHARACTER	Date and time the FileRead node started processing the file as a character string, in the Coordinated Universal Time (UTC) zone.
The following elements contain data about the current record:		
Offset	INTEGER	The offset in the file the record starts at. The first byte in the file is offset 0.
NextRecordOffset	INTEGER	The offset in the file that the next record starts at, relative to the start of the file, and is 1 byte after the end of the current record. If the end of the file is reached, then the value is not given in the local environment.
EndOfFile	BOOLEAN	The FileRead node sets this element to TRUE when it has read the last record of the input file. It is therefore always TRUE when the detection property is `Record is Whole File`.
RecordNumber	INTEGER	The number of the record in the file relative to the offset the read node starts reading from. The value is always 1 unless the filter expression is being used, in which case it reflects the number of the record that was selected.
NoMatchReason	STRING	The reason why a message is sent to the "No match" terminal. Null if the message is sent to the Out terminal. Possible reasons: • NoFile - the file does not exist. • NoData - the file exists but has no records. • NoRecord - the file exists and contains records but none match the filter expression.

Element Name	Element Data Type	Description
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. If this element is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. It is set to TRUE if the current record is empty.
Archive/Directory	STRING	The name of the directory where the file was archived.
Archive/Name	STRING	The name of the file where the file was archived.
TransferredFile.FileName	CHARACTER	If a file is transferred from a remote server, this field contains the name of the file.
TransferredFile.ServerDirectory	CHARACTER	If a file is transferred from a remote server, this field contains the directory on the remote server that the file was transferred from.
TransferredFile.LocalDirectory	CHARACTER	If a file is transferred from a remote server, this field contains the integration node's local file system directory that the file was transferred to.
TransferredFile.TimeStamp	GMTTIMESTAMP	If a file is transferred from a remote server, this field contains a time stamp that represents the time that the file was transferred.

This structure is propagated with each message written to the Out terminal of the FileRead node and with the empty message written to the End of data terminal.

LocalEnvironment.WrittenDestination.File fields

When you use the FileOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.File message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Action	CHARACTER	Possible values are: • `Replace` if an output file of the same name is replaced. • `Create` if a new output file is created. • `Append` if this value is associated with a record that is appended to an output file. • `Finish` if a Finish File message is received and no file is found to finish (for example, if `Record is Whole File` is specified and a message is sent to the Finish File terminal). • `Transmit` if the file was transferred by FTP or SFTP and the file was not retained.

Element Name	Element Data Type	Description
Timestamp	CHARACTER	The date and time, in character string form, when the node started to process this file. This value prefixes the names of files that are archived if you specify `Time Stamp, Archive, Replace Existing File and Append to Existing File` in the `Action if file exists` property on the **Basic** tab.

LocalEnvironment.Destination.File fields

When you use the FileOutput and FileRead nodes, you can override the directory and name properties with elements in the message tree. The default location for these overrides is LocalEnvironment.Destination.File, although you can change this location by using the properties on the `Request directory property location` and `Request file name property location` on the FileOutput node. When you use the FileRead node, you can also override the length and offset properties. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	This property specifies the absolute or relative directory path of the output directory in the form that is used by the file system of the integration node. For example, on Windows systems, this path starts with the drive letter prefix (such as C:) and use a backslash (\) as the directory delimiter. On UNIX systems, the path includes a slash (/) as the directory delimiter.
Name	CHARACTER	This property specifies the file name of the output file. The FileOutput node does not perform wildcard replacement on the value of the element. For example, if its value is `Input*.txt`, the FileOutput node tries to write to a file with an asterisk (*) in its name. It might or might not succeed, depending on whether an asterisk is a valid character for files in the file system to which it is writing.
Length	INTEGER	This property specifies the length of the record to read from the file. The value is only used if the record detection option fixed length is being used.
Offset	INTEGER	This property specifies the offset in the file to start searching for a record. Offset 0 means start from the beginning and is the default value if no override is given.
Archive/Directory	STRING	The directory where the file is archived to when using one of the file disposition archive options. By default the file is archived to 'mqsiarchive' under the file input directory. Any path is not relative to the input directory but relative to the `MQSI_FILENODES_ROOT_DIRECTORY`.
Archive/Name	STRING	The pattern to use to create an archive file name. Only one star is allowed in the file name and the star is replaced with the first star replace in the file pattern name. If `Archive with Time Stamp` is specified, then a time stamp is appended to the archive name.
PosixPermissions	CHARACTER	A 3-digit octal string that specifies the permissions that are required on the output file. This field uses the same syntax as the syntax that is used in the **chmod** command; for example, a value of 666 corresponds to **rw-rw-rw**.

LocalEnvironment.Destination.File.Remote fields

When you use the FileOutput node or the FileRead node with the `Remote Transfer` property selected, you can override the directory name with an element in the local environment tree. When you use the FileRead node, you can override the remote server with an element in the local environment tree. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
Remote.ServerDirectory	CHARACTER	This property specifies the absolute or relative directory path of the output directory on the remote server. The property has no effect if FTP or SFTP are not enabled on the FileOutput node. Format the path according to the path syntax that is accepted by the FTP server, typically by using UNIX-style slash (/) directory delimiters. This property is supported by the FileOutput and FileRead nodes.
Remote.Server	CHARACTER	This property can contain either the name of an FTP Server policy, or a string of the form `hostname:port`. If you set a policy name, all properties that are set to non-default values on the policy are used in preference to the properties that are defined on the node. For a `hostname:port` string, these values are used when connecting to a remote FTP or SFTP server.

LocalEnvironment.Wildcard.WildcardMatch field

On the FileInput, CDInput, and FTEInput nodes, you can specify a file name pattern that contains wildcard characters. The input nodes copy the characters in the file name matched by wildcards, together with any intermediate characters, to LocalEnvironment.Wildcard.WildcardMatch.

Element Name	Element Data Type	Description
WildcardMatch	CHARACTER	The character string in the file name matched by wildcards in the file name pattern.

On the FileOutput, CDOutput, and FTEOutput nodes, you can use a wildcard character in the file name pattern. If you include the single wildcard character, '*', in the file name pattern, the node uses the value that is stored in LocalEnvironment.Wildcard.WildcardMatch. This is useful if you have a message flow where the input and output nodes are working with the same file; you can preserve the name of the input file on the output nodes. You can also use standard methods for manipulating the value of the WildcardMatch element to whatever you want; you must not use a FileInput, CDInput, or FTEInput node.

See “File name patterns” on page 972 for more information.

LocalEnvironment.FTE fields

When you use the FTEInput node, it stores information that you can access in the LocalEnvironment.FTE and LocalEnvironment.FTE.Transfer message trees. The LocalEnvironment.FTE message tree stores information relating to the current record and is populated by the integration node. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
TimeStamp	CHARACTER	Date and time the input node started processing the file in the Coordinated Universal Time (UTC) zone, as a character string. This data is the string used to create archive and backout file names if a timestamp is included.
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0 bytes. When Offset is part of the End of Data message tree, this value is the length of the input file.

Element Name	Element Data Type	Description
Record	INTEGER	Number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this value is always set to TRUE.

The LocalEnvironment.FTE.Transfer message tree contains information received from IBM MQ File Transfer Edition regarding the transfer or file; see WebSphere File Transfer Edition product documentation for more details. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
Directory	CHARACTER	Absolute directory path of the input directory.
JobName	CHARACTER	The name for the transfer.
Name	CHARACTER	File name and extension (per file).
LastModified	TIMESTAMP	Date and time the file was last modified (per file).
SourceAgent	CHARACTER	The name of the agent sending the file.
DestinationAgent	CHARACTER	The name of the agent to send the file to.
OriginatingHost	CHARACTER	The name of the host from which the transfer was submitted.
TransferId	CHARACTER	The unique name of the transfer.
MQMDUser	CHARACTER	The IBM MQ user ID in the MQMD of the transfer request message.
OriginatingUser	CHARACTER	The user ID of the user that submitted the transfer request.
TransferMode	CHARACTER	The mode of the transfer. Valid values are `Binary` or `Text`.
TransferStatus	CHARACTER	The status of the transfer of the file.
FileSize	INTEGER	The size of the file being transferred.
ChecksumMethod	CHARACTER	The only allowed value is `MD5`.
Checksum	CHARACTER	If the ChecksumMethod element is set to `MD5`, this element is the actual checksum in hex string format.
DestinationAgentQmgr	CHARACTER	The name of the queue manager of the destination agent to send the file to.
SourceAgentQmgr	CHARACTER	The name of the queue manager of the source agent that sent the file.
OverallTransferStatus	CHARACTER	The overall status of the transfer.
TotalTransfers	INTEGER	The total number of files successfully transferred.
TransferNumber	INTEGER	The number of the current file in the transfer.

These structures are propagated with each message written to the Out terminal of the FTEInput node and with the empty message written to the End of data terminal.

LocalEnvironment.WrittenDestination.FTE fields

When you use the FTEOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.FTE message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
DestinationAgent	CHARACTER	The name of the agent to send the file to.
DestinationQmgr	CHARACTER	The name of the destination queue manager.
JobName	CHARACTER	The name for the transfer.
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Overwrite	BOOLEAN	Specifies whether files on the destination system can be overwritten when the destination agent moves files of the same name there. If the destination agent fails to overwrite the file, the transfer fails and the transfer logs report the failure. The FTEOutput node does not throw or log any errors.
TransferId	CHARACTER	The unique name of the transfer initiated by the FTEOutput node.

LocalEnvironment.Destination.FTE fields

When you use the FTEOutput node, you can override its Destination agent, Destination queue manager, Job name, Destination file directory, Destination file name, and Overwrite files on destination system properties with elements in the message tree. You can also call a program on the destination agent before starting the transfer, or when the transfer is finished. The default location for these overrides is LocalEnvironment.Destination.FTE. The fields of this structure are described in the following table.

Element Name	Element Data Type	Description
DestinationAgent	CHARACTER	The name of the agent to send the file to.
DestinationQmgr	CHARACTER	The name of the destination queue manager.
JobName	CHARACTER	The name for the transfer.
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Overwrite	BOOLEAN	Specifies whether files on the destination system can be overwritten when the destination agent moves files of the same name there. If the destination agent fails to overwrite the file, the transfer fails and the transfer logs report the failure. The FTEOutput node does not throw or log any errors.

Element Name	Element Data Type	Description
PreDestinationCall.Name	CHARACTER	Call a program on the destination agent before starting the transfer. This element supplies the name of an Ant script to run. The Ant script can access all the metadata that is defined for the transfer, including user metadata added by using the local environment override `LocalEnvironment.Destination.FTE.UserDefined`. You must be aware of the following restrictions: • You cannot call other programs that are not Ant scripts, or pass parameters to the calls. • The destination agent cannot be an FTE agent that is embedded in an integration server process. See WebSphere File Transfer Edition product documentation for more details of how to use the PreDestinationCall function.
PostDestinationCall.Name	CHARACTER	Call a program on the destination agent after completing the transfer. This element supplies the name of an Ant script to run. The Ant script can access all the metadata that is defined for the transfer, including user metadata added by using the local environment override `LocalEnvironment.Destination.FTE.UserDefined`. You must be aware of the following restrictions: • You cannot call other programs that are not Ant scripts, or pass parameters to the calls. • The destination agent cannot be an FTE agent that is embedded in an integration server process. See WebSphere File Transfer Edition product documentation for more details of how to use the PostDestinationCall function.

LocalEnvironment.CD fields

When you use the CDInput node, it stores information that you can access in the LocalEnvironment.CD and LocalEnvironment.CD.Transfer message trees. The LocalEnvironment.CD message tree stores information relating to the current record and is populated by the integration node. The fields in this structure are described in the following table:

Element Name	Element Data Type	Description
Transfer	Folder	Contains meta data from the IBM Sterling Connect:Direct transfer.
Timestamp	CHAR	Timestamp of the file.
Offset	INTEGER	Start of the record within the file. The first record starts at offset 0 bytes. When Offset is part of the End of Data message tree, this value is the length of the input file.
Record	INTEGER	Number of the record within the file. The first record is record number 1. When Record is part of the End of Data message tree, this value is the number of records.

Element Name	Element Data Type	Description
Delimiter	CHARACTER	The characters used to separate this record from the preceding record, if `Delimited` is specified in `Record detection`. The first record has a null delimiter. When Delimiter is part of the End of Data message tree, this value is the delimiter that follows the last record, if any.
IsEmpty	BOOLEAN	Whether the record propagated by the message flow is empty. IsEmpty is set to TRUE if the current record is empty. When IsEmpty is part of the End of Data message tree, this value is always set to TRUE.

These structures are propagated with each message written to the Out terminal of the CDInput node and with the empty message written to the End of data terminal.

LocalEnvironment.CD.Transfer

The LocalEnvironment.CD.Transfer message tree contains information received from IBM Sterling Connect:Direct regarding the transfer or file. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
ProcessName	CHARACTER	The process name of the script transferring the file.
StepName	CHARACTER	The step name causing the transfer to take place.
ProcessNumber	INTEGER	The number of the process running the process script.
Submitter	CHAR	The user ID submitting the process script.
Accounting	CHAR	The secondary node (SNODE) accounting details for the process script.
SourcePath	CHAR	The source path of the file on the primary node (PNODE) machine.
DestinationPath	CHAR	The destination path of the file on the secondary node (SNODE) machine
Directory	CHARACTER	The directory which the file is copied to.
Name	CHARACTER	The name of the file copied to.
PrimaryNodeName	CHARACTER	The name of the primary node from which the file was copied.
SecondaryNodeName	CHARACTER	The name of the secondary node.

LocalEnvironment.Destination.CD fields

When you use the CDOutput node, you can override various destination system properties with elements in the message tree. The default location for these overrides is LocalEnvironment.Destination.CD. The fields of this structure are described in the following table

Element Name	Element Data Type	Description
SNODE	CHARACTER	The name of the secondary Connect:Direct server (SNODE) to send the file to.
SNODEID	CHARACTER	The user ID and password of the secondary Connect:Direct server (SNODEID) to send the file to.
ProcessName	CHARACTER	The process name that the script uses to run.

Element Name	Element Data Type	Description
Accounting	CHARACTER	Accounting data shown when the script is running on both the primary Connect:Direct server (PNODE) and secondary Connect:Direct server (SNODE).
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
Copy.From	CHARACTER	The final part of the path name is the IBM Sterling Connect:Direct process script property you want to change. This is either a direct `<option name>` on the FROM clause, or a value in the <SYSOPTS> option. You must take care to ensure that the created script is valid, because any existing value created by the node is overridden.
Copy.To	CHARACTER	The final part of the path name is the IBM Sterling Connect:Direct process script property you want to change. This is either a direct `<option name>` on the TO clause, or a value in the <SYSOPTS> option. You must take care to ensure that the created script is valid, because any existing value created by the node is overridden.

LocalEnvironment.WrittenDestination.CD fields

When you use the CDOutput node, it stores information that you can access in the LocalEnvironment.WrittenDestination.CD message tree. The fields in this structure are described in the following table.

Element Name	Element Data Type	Description
ProcessName	CHARACTER	The name of the process sending the file.
ProcessNumber	CHARACTER	The number of the process sending the file.
Directory	CHARACTER	Absolute directory path of the output directory in the form used by the file system of the integration node. For example, on Windows systems, this starts with the drive letter prefix (such as `C:`).
Name	CHARACTER	File name of the output file.
PrimaryNodeName	CHARACTER	The name of the primary Connect:Direct server (PNODE)
PrimaryNodeOS	CHARACTER	The operating system of the primary Connect:Direct server
SecondaryNodeName	CHARACTER	The name of the secondary Connect:Direct server (SNODE)
SecondaryNodeOS	CHARACTER	The operating system of the secondary Connect:Direct server (this might not be the same as the IBM App Connect Enterprise operating system)

File name patterns

You can specify a file name pattern, using wildcard characters, to identify a file to be read by the FileInput, CDInput, and FTEInput nodes. You can also specify a file name pattern, using a single wildcard character, to name the file to be created by the FileOutput and FTEOutput nodes.

Using file name patterns with the FileInput, CDInput, and FTEInput nodes

The input nodes read files from a specified directory and propagate messages based on the contents of these files. Only files with names that match a pattern (the input pattern), as specified in the FileInput node's `File name or pattern` property or the FTEInput node's `File name filter` property, are read.

If you are using file name patterns to exclude files from the FileInput node, files with names that match the exclude pattern, as specified in the FileInput node's `File exclusion pattern` are not read or processed.

The match might be with a file name or a character sequence (a pattern). A pattern is a sequence containing at least one of the following wildcard characters:

Wildcard character	Description	Example
*	Any sequence of zero or more characters	`*.xml` matches all file names with an `xml` extension
?	Any single character	`f??????.csv` matches all file names consisting of the letter `f` followed by six characters, then the sequence `.csv`.

The default pattern is `*`, which matches all file names.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), or file name separator characters ('/' and' \').

For example:

- If you want the FileInput node to process all files that have a certain extension, such as `xml`, set its `File name or pattern` property to `*.xml` and the node will process all files in the directory (and subdirectories if selected) that have this extension.
- If you want the FileInput node to exclude all files with a certain extension from being processed, such as `.csv`, set its `File exclusion pattern` property to `*.csv` and the node will not process any of the files with this extension in the directory (and subdirectories if selected).

If you deploy the flow to a Windows server, file names match the pattern irrespective of case. However, if you deploy the flow to a Linux, UNIX, or z/OS server, file names must match the pattern character string and its case.

Pattern matching

The FileInput, CDInput, and FTEInput nodes set the LocalEnvironment.Wildcard.WildcardMatch element to the string matched by wildcards in the file name. The LocalEnvironment.Wildcard.WildcardMatch element is unaffected by file exclusion patterns. The following are some examples of pattern matching with the value in this element, where the value in the FileInput node's `File name or pattern` property is `File????.from*.xml`:

- If the FileInput node finds a file with the file name `File1234.fromHQ.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `1234.fromHQ` and the node processes the file.
- If the file name is `File123.fromHQ.xml`, there is no match because there are insufficient characters between the `File` and `.from` elements of the file name. The FileInput node ignores this file.

- If the file name is `File2345.from.xml`, there is a match. The value in the LocalEnvironment.Wildcard.WildcardMatch element is set to `2345.from` and the node processes the file. In this example, the `*` in the character string in the `File name or pattern` property matches a string of zero characters. If you require the character string between the `from` and `.xml` elements of the file name to always have at least one character, you specify the `File name or pattern` property with a value of `File????.from?*.xml`.

Using file name patterns with the FileOutput, CDOutput, and FTEOutput nodes

The node writes messages to files that it creates or replaces in the integration node's file system. Only patterns containing a single wildcard character (the asterisk, '*') are allowed in this property. The file name to be used is determined in the following way:

- If the file name property contains no wildcard, the value of this property is the name of the file created. This value must be a valid file name on the file system that hosts the integration node to which the message flow is deployed.
- If the file name property contains a single wildcard, the value of the element LocalEnvironment.Wildcard.WildcardMatch in the current message replaces the wildcard character, and the resulting value is the name of the file created. This value must be a valid file name on the file system that hosts the integration node to which the message flow is deployed. If the WildcardMatch value is not found, the wildcard character is replaced by the empty string.

You cannot specify file names that contain the following characters: the asterisk ('*'), the question mark ('?'), file name separator characters ('/' and' \'). The name of the file can be overridden by values in the current message.

If the `File name or pattern` property on the FileOutput node is empty, the name must be overridden by the current message. Wildcard substitution occurs only if this property is not overridden in this way.

File names are passed to the file system to which the integration node has access, and have to respect the conventions of these file systems. For example, file names on Windows systems are not case-sensitive; on UNIX systems, file names that differ by case are considered distinct.

Example: If the FileInput node has `*.out` in the `File name or pattern` property, and the incoming file is `myfile`, the name of the outgoing file is `myfile.out`.

FTP and SFTP considerations

You can use the FileInput node to transfer files from a remote FTP or SFTP server and process them. Only files with names that match the file name pattern specified in the node are read. You can also use a file exclusion pattern to prevent files from being read on a remote FTP or SFTP server. If your integration node is on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. If you then use this pattern to process files that are in a directory on a remote FTP or SFTP server, and this server is running on an operating system that does not respect case sensitivity (such as Windows), file name matching might fail with the result that no files are processed. This failure occurs because the file names on the remote server are not in mixed case. If your integration node is on an operating system that does not respect case sensitivity, any pattern that you specify might be matched by more than one file on a remote FTP or SFTP server that is running on an operating system on which case sensitivity is significant. Each of these files is then processed sequentially.

You can use the FileOutput node to write files to a remote FTP or SFTP server. Only files with names that match the pattern specified in the node are written. If your integration node is running on an operating system that respects case sensitivity (such as UNIX), you might specify a pattern that includes a combination of uppercase and lowercase characters. However, if you then use this pattern to write files to a directory on a remote FTP or SFTP server running on an operating system that does not respect case sensitivity (such as Windows), the file name is written in uppercase rather than in the way that it was specified in your pattern.

If the name of a file on a remote FTP server contains one or more characters that are not valid on the operating system on which the integration node where you specified the file name pattern is running, the file is not transferred from the FTP server for processing by the FileInput node.

Archiving

Files that are successfully processed by the FileInput node or FileOutput node can optionally be moved to the `mqsiarchive` subdirectory of the input or output directory.

The input directory of the FileInput node has a subdirectory called `mqsiarchive`. The output directory of the FileOutput node also has a subdirectory called `mqsiarchive`. Archiving is not enabled for the FTEOutput and FTEInput nodes.

FileInput node

Files that are processed successfully by the FileInput node are moved to the `mqsiarchive` subdirectory if the FileInput node's `Action on successful processing` property is set to `Move to Archive Subdirectory` or `Add Timestamp and Move to Archive Subdirectory`.

Select the `Replace duplicate archive files` check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node stops processing files. Every time that the node returns from its polling wait period, it issues a pair of messages, BIP3331 and a more specific one describing the problem. To avoid writing too many messages, duplicate messages are suppressed for increasing periods of time, until eventually they are issued only about once every hour. In this circumstance, the system administrator must stop the flow, correct the problem, then restart the flow.

Clear the `Replace duplicate archive files` check box only if you are sure either that the input files have unique names, or that some other process will remove a file from the archive directory before the FileInput node processes another of the same name. If you cannot ensure this, either specify `Add Timestamp and Move to Archive Subdirectory` in the `Action on successful processing` property so that archived files have unique names, or select the `Replace duplicate archive files` check box

FileOutput node

Files that are processed successfully by the FileOutput node are moved to the `mqsiarchive` subdirectory if the `Action if file exists` property of the FileOutput node is set to `Append to Existing File`, `Archive and Replace Existing File` or `Time Stamp, Archive and Replace Existing File`.

Select the `Replace duplicate archive files` check box to overwrite existing files in the `mqsiarchive` subdirectory. If you do not set this option, and a file with the same name already exists in the archive subdirectory, the node generates an exception when it tries to move the successfully processed file and the file remains in the transit subdirectory.

Reading files

Use the FileInput, CDInput, FTEInput, and FileRead nodes to read files.

About this task

This section contains the following topics:

Using a local file as input for your message flow
Learn how to use the FileInput node to read a file on your local file system and then propagate messages that are based on the contents of that file.

Before you begin

This example shows how one combination of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties can be used to extract messages from a file. The example describes the FileInput node of a message flow and assumes that the rest of the flow has already been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have added a FileInput node to a message flow. You also need the following resources:

- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

  Each line ends with a line terminator; on a Windows system, this comprises carriage return and line feed characters (X'0D0A'). Put this file into directory `C:\FileInput\TestDir`.
- A message set. This example uses a message set called `xml1` which uses the XMLNSC parser. Message set `xml1` models messages of the following form:

  ```
  <Message>...</Message>
  ```

For more information about configuring the FileInput node, see Configuring the node.

Complete the following steps:

Procedure

1. Set the required node properties on the FileInput node.

   The following table summarizes the FileInput node properties that you should set, which tab they appear on and the value that you should set in order to follow this example:

Tab	Property	Value
Basic	Input directory	C:\FileInput\TestDir
	File name or pattern	test_input1.xml
	Action on successful processing	Move to Archive Subdirectory
	Replace duplicate archive files	Selected
Input Message Parsing	Message domain	XMLNSC
	Message model	xml1
Polling	Polling interval	3
Retry	Action on failing file	Add Time Stamp and Move to Backout Subdirectory
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix
FTP	FTP	Not selected

2. Deploy the message flow to the integration node. See Chapter 7, “Deploying integration solutions,” on page 2463.

Results

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties on the **Records and Elements** tab, the FileInput node detects records that are separated by DOS or UNIX end-of-line characters and creates a message for each one that it finds. It propagates three messages to the flow attached to the Out terminal:
   - Message 1:

     ```
     <Message>test1</Message>
     ```

   - Message 2:

     ```
     <Message>testtwo</Message>
     ```

   - Message 3:

     ```
     <Message>testthree</Message>
     ```

2. If a flow is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.
3. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory, `C:\FileInput\TestDir\mqsiarchive\test_input1.xml`. If a file called `test_input1.xml` already exists in the `mqsiarchive` subdirectory, it is overwritten.
4. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput\TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

What to do next

To see the effects of specifying other combinations of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties of the FileInput node, see “Controlling how files are separated into records” on page 985.

To extend this example to process files from within subdirectories, put the input file in a subdirectory of `Input directory`, and select the property `Include local subdirectories`.

Routing or enriching a message based on the contents of a file

The FileRead node can route or enrich messages based on the contents of the file.

When developing the message flow you can specify the name and location of the file to be read. You can override these values at run time based on the contents of a message.

The node complements the existing FileInput and FileOutput nodes. The FileRead node reads a file in the middle of a message flow.

Using the node to route messages

A message is routed by using the contents of a file colocated with IBM App Connect Enterprise or on a network file system. The message from the source system is routed to a target system by using an external routing file. No response is expected.

The basic flow of events is as follows:

- IBM App Connect Enterprise receives a message through an input node.

- The IBM App Connect Enterprise message flow interrogates the contents of a message to identify routing key information.
- If the file consists of more than one record, you need to determine:
  - Where the first record starts. Unless you specify an offset byte, the node starts reading the file at the first byte.
  - How each record ends (fixed-sized, delimited, or parsed.)
  - Which record to propagate. You can use any combination of information from the input message and the file in deciding this. All records from the specified start point are read until a record is found that matches the record selection expression, this record is then propagated. Examples include:

    The third record, as identified by the local environment field `$OutputLocalEnvironment/File/Read/RecordNumber=3`. In this example, the first record is fully read and the expression evaluates to false. The second record is then fully read and the expression evaluates to false. When the third record is fully read, the expression evaluates to true and the record is propagated. No further records are read.

    A key field in the input message matches a key field in the file `$InputRoot/XMLNSC/FromMQInputMessage/Record1 = $ResultRoot/XMLNSC/FromFile/Record5`. In this example, records are read from the file until the value of the Record5 element of the record matches the value of the Record1 element of the incoming message. The location of the record in the file determines how many records the node must read before successfully matching the record selection expression.
- Within the message flow you can implement a local cache of records to reduce the performance cost of reading multiple static records.
- You can choose to take information from the file, and copy it to the outgoing message. The copy can be a subset of the data, and can be copied to any location in the message or the local environment. For more information, see "Combining a result message with an input message when fetching data from external systems" on page 1811.
- A target application receives the routed message.

Combining an IBM MQ message with an XML file using the contents of the message to identify which file to use
Combine an incoming message with the contents of an XML file, using fields in the message to determine which file to use.

Before you begin

Put a file on the file system that is local to the integration node, for the FileRead node to read. Here is an example of the file contents:

```
<Data>Purchase details</Data>
```

In this example, the contents of the data in the data tag are inserted into the incoming message. Any valid XML structures can be added to this section.

Make a note of the path to the file. For example: `c:\temp\FileRead\task3.xml` or `/tmp/FileRead/task3`.

Create the following queues on the integration node queue manager:

- FILEREAD.TASK3.IN1
- FILEREAD.TASK3.OUT1

Detailed information about configuring the node is given on the property panels for the node, in the IBM App Connect Enterprise Toolkit.

Procedure

1. Create a message flow that contains an MQInput node, a FileRead node, and an MQOutput node.

2. Wire the terminals as follows:
   a. Wire the Out terminal of the MQInput node to the In terminal of the FileRead node.
   b. Wire the Out terminal of the FileRead to the In terminal of the MQOutput node.
3. Configure the MQInput node:
   a. On the Basic panel, set the Queue name to `FILEREAD.TASK3.IN1`
   b. On the Input Message Parsing panel set the domain to `XMLNSC`.
4. Configure the FileRead node.
   a. On the Basic panel, set the directory and file name to refer to the XML file. For example: `c:\temp\FileRead` and `task3.xml` or `/tmp/FileRead` and `task3`.
   b. Configure the Result panel:
      i) Set the Result data location to `$ResultRoot/XMLNSC/Data`
      ii) Set the Output data location to `$InputRoot/XMLNSC/Data`
   c. Configure the Input Message Parsing panel:
      i) Set the Domain to `XMLNSC`
5. Configure the MQOutput node:
   a. On the Basic panel, set the Queue name to `FILEREAD.TASK3.OUT1`
6. Deploy the message set and message flow.
7. Change the Directory and Name fields to the correct location of the file, and then put the following XML message onto queue FILEREAD.TASK2.IN1:

```
<Invoice>
  <Directory>c:\temp\FileRead</Directory>
  <Name>task2.xml</Name>
  <Data/>
</Invoice>
```

Results

The integration node routes the message to the queue FILEREAD.TASK3.OUT1 and inserts data from the file into the Data field of the output message:

```
<Invoice>
  <Directory>c:\temp\FileRead</Directory>
  <Name>task2.xml</Name>
  <Data>Purchase details</Data>
</Invoice>
```

Record selection expressions
The record selection expression is used to select a record from a file to propagate to the rest of the flow.

Each record in turn is compared to the expression, and the first one to evaluate to true is propagated to the **Out** terminal. You can set the expression to any valid XPath expression that returns a Boolean value. The expression is not used when `Whole File` is selected as the `Record Detection` option.

Correlation names used in the expression

You can use any of the following correlation names in the expression:

`InputRoot` and `InputLocalEnvironment`
The Input names refer to the incoming message that enters the node through the **In** terminal

`ResultRoot`
The `ResultRoot` name refers to the message created by using the current record in the file.

OutputRoot
The OutputRoot name refers to the message that is propagated if the expression evaluates to true. This action is identical to ResultRoot, unless the Output data location or Result data location have been changed to copy the Result message found in the file to a different location in the outgoing message.

OutputLocalEnvironment
The OutputLocalEnvironment contains the normal local environment which is propagated down the **Out** terminal and contains useful information like the number of the record and its offset.

Any combination of data in these correlation names can be used, along with any valid XPath expression, to determine whether to propagate the record.

Examples

Expression is:

```
$InputRoot/XMLNSC/Invoice/AccountNumber=$ResultRoot/XMLNSC/Data/Key
```

In this example, each record is a valid XML document. The FileRead node reads each record from the file. In the incoming message the FileRead node compares the field /Data/Key to the field /Invoice/AccountNumber. If the record matches, it is propagated to the **Out** terminal.

Expression is:

```
$OutputLocalEnvironment/File/Read/RecordNumber=5
```

The FileRead node reads each record from the file and compares the record number to 5. The record is propagated when it reaches the fifth record.

Building an outgoing message by using an incoming message combined with a record from a file

The FileRead node reads a record from a file and combines it with the message coming in to the node. By default, it replaces the message with the contents of the record read from the file. However, by using properties on the Result panel, you can choose how to combine the incoming message and file record contents. The node has three logical trees:

Input
The Input message assembly contains all the data in the incoming message and is the basis for the propagated record.

Result
The Result message assembly contains the record read from the file.

Output
The Output message assembly is the actual object propagated from the node.

By default, the Output message assembly is constructed by copying the Input message assembly to the Output message assembly. The data part of the Output message assembly is then replaced by the contents of the Result message assembly and the OutputLocalEnvironment is updated with details of what happened in the node.

The following Result panel properties can be used to modify this behavior:

Result data location
Specifies which part of the read record is copied to the Output message. By default, the Result data location copies everything from ResultRoot but it can be changed to copy only part of the record. For example: ResultRoot.XMLNSC.Invoice.Name just copies the name field from the selected record to the output message.

Output data location
Specifies where the record is copied to in the outgoing message. By default, the Output data location copies everything to OutputRoot. The location specified can be in the data part of the message (under

ResultRoot) or in any other Output tree like OutputLocalEnvironment. For example: To copy the resulting record to a field in the message body OutputRoot.XMLNSC.Invoice.Data or to copy the result to local environment OutputRoot.Variables.Invoice.data.

Copy local environment
Causes the local environment to be copied from the InputLocalEnvironment. If the Copy local environment option is not selected, the InputLocalEnvironment is used directly without copying. This option allows nodes before the FileRead node to see changes to the Local environment.

For example, the following options copy the name field from a record to the output going message. The rest of the Output message is based on the input message:

```
Result data location= ResultRoot.XMLNSC.Invoice.Name
Output data location= OutputRoot.XMLNSC.Invoice.Name
```

The following options copy the message body from a record to the output going local environment. The Output message is the same as the input message:

```
Result data location= ResultRoot.XMLNSC.Invoice
Output data location= OutputLocalEnvironment.Variables.Invoice
```

Reading a file on a remote FTP or SFTP directory
Use a FileInput node to read a file in a directory on a remote FTP or SFTP server and then propagate messages that are based on the contents of that file.

Before you begin

This example is an extension of the example described in "Using a local file as input for your message flow" on page 975 and it describes how to use a FileInput node in a message flow. The instructions assume that you are using a Windows operating system and that you have created a message flow containing a FileInput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists, with the following settings:

  Server
  ftpserver.hursley.abc.com

  Port
  21 (for FTP) or *22* (for SFTP)

  Working directory
  /ftpfileinput

  Userid
  myuserid

  Password
  mypassword

  These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values during the task.

- A security identity. Use the **mqsisetdbparms** command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example, use the following command:

  ```
  mqsisetdbparms workDir -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms -w workDir -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use public key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms -w workDir -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  To check what security credentials are already set, use the **mqsireportdbparms** command; see "Checking the password for a resource that is used by an integration server" on page 2630.

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 998.
- An input file. To follow this example scenario, create an input file called `test_input1.xml` with the following content:

  ```
  <Message>test1</Message>
  <Message>testtwo</Message>
  <Message>testthree</Message>
  ```

  Each line ends with a line terminator that is suitable for the system on which the FTP or SFTP server is found. Do not put this file in the input directory but, instead, put it in the FTP or SFTP server directory `/ftpfileinput`.
- A message set. This example uses a message set called *xml1*, which uses the XMLNSC parser. Message set *xml1* models messages of the following form:

  ```
  <Message>...</Message>
  ```

About this task

Complete the following steps:

Procedure

1. Set the required node properties on the FileInput node.

   The following table summarizes the FileInput node properties that you must set, the tab on which they are displayed, whether they are mandatory, and the required values.

Tab	Property	Value
Basic	`Input directory`	`C:\FileInput\TestDir` If the input directory does not exist, no files are processed, even if you are processing files over FTP or SFTP.
	`File name or pattern`	`test_input1.xml`
	`Action on successful processing`	`Move to Archive Subdirectory`
	`Replace duplicate archive files`	Selected

Tab	Property	Value
Input Message Parsing	Message domain	XMLNSC
	Message model	xml1
Polling	Polling interval	3
Retry	Action on failing file	Add Time Stamp and Move to Backout Subdirectory
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix
FTP	Remote transfer	Selected
	Transfer protocol	FTP or SFTP
	Remote server and port	ftpserver.hursley.abc.com
	Security identity	myidentity
	Server directory	/ftpfileinput
	Transfer mode	ASCII (for FTP only)
	Scan delay	45

If you used other values for your FTP or SFTP server resource, enter those values. The settings used here are identical to those used in the example in "Using a local file as input for your message flow" on page 975, except that the Remote transfer property has been selected and there are now properties on the **FTP** tab. If you clear the Remote transfer property, the node operates as it does in the example in "Using a local file as input for your message flow" on page 975; the properties on the **FTP** tab remain set but are ignored.

2. Deploy the message flow to the integration server. See Chapter 7, "Deploying integration solutions," on page 2463.

Results

The following actions occur when you perform these steps:

1. The file test_input1.xml is transferred from the FTP or SFTP server directory (/ftpfileinput) to the local directory (C:\FileInput\TestDir). The file is deleted from the FTP or SFTP server directory.
2. The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds, as defined by the properties on the **Records and elements** tab. The node propagates three messages to the message flow that is attached to the Out terminal:
   - Message 1:

     ```
     <Message>test1</Message>
     ```

   - Message 2:

     ```
     <Message>testtwo</Message>
     ```

   - Message 3:

     ```
     <Message>testthree</Message>
     ```

3. If a node is attached to the End of Data terminal, the End of Data message is propagated after the last record in the file has been processed.

4. When processing is complete, the file `test_input1.xml` is moved to the `mqsiarchive` subdirectory `C:\FileInput\TestDir\mqsiarchive`. If a file called `test_input1.xml` exists in the `mqsiarchive` subdirectory, it is overwritten.
5. If the message flow fails, retry processing is attempted according to the values set in the properties of the FileInput node. In this example task, a time stamp is added to the file name and the file is moved to the `mqsibackout` directory. Here is an example of the path to such a file: `C:\FileInput \TestDir\mqsibackout\20070928_150234_171021_test_input1.xml`.

   If an error occurs on the FTP side, stating that access is denied, a 0-byte file is created and moved to the mqsibackout directory. A 0-byte file is created in the mqsibackout directory for every FTP attempt that fails.

Because the `Remote transfer` property is selected, the FTP scan delay of 45 seconds overrides the polling interval of 3 seconds.

What to do next

For more information, see "Controlling how files are separated into records" on page 985, which shows the effects of specifying other combinations of values in the `Record detection`, `Delimiter`, and `Delimiter type` properties of the FileInput node.

Receiving a file by IBM MQ File Transfer Edition
Use the FTEInput node to receive files from an existing IBM MQ File Transfer Edition network.

Before you begin

1. Read about IBM MQ File Transfer Edition.
2. Ensure that IBM MQ is installed on the same machine as IBM App Connect Enterprise. Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the FTEInput node.
3. Create the system queues that will store the file transfer information. For more information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.

About this task

Complete the following steps to create an FTEInput node message flow using the default settings. The task includes some optional steps for additional configuration, but detailed information about configuring the FTEInput node is given on the property panels for the node, in the IBM App Connect Enterprise Toolkit.

Procedure

1. Drag an FTEInput node onto a message flow and wire its Out terminal to an output node of your choice.
2. Optional: To process only a defined subset of files sent to an agent, configure the **Basic panel**.

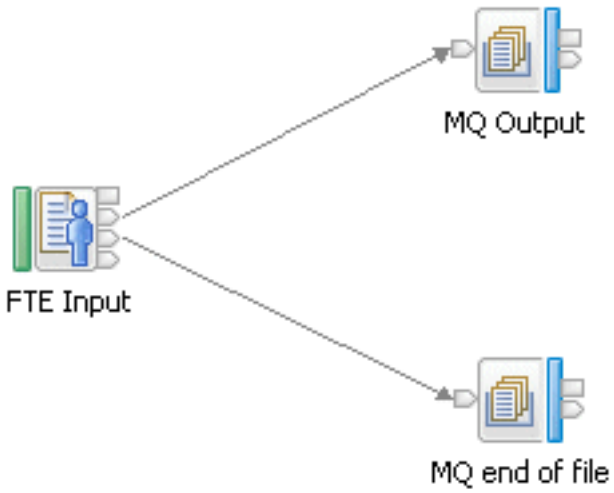
   This action allows multiple FTEInput nodes in the same integration server to receive specific files, depending on the directory or file filters specified.

   You can also specify whether, after processing, the file should be left in its directory, renamed, or deleted.
3. To change how the node handles a message flow failure, configure the Retry panel.

4. Optional: To change how records are identified in the input file, configure the `Record detection` property on the **Records and Elements** panel.

   For example, you might want to specify that a record is fixed length, and set the record length.

   a) If you set the `Record detection` property to anything other than `Whole File`, drag an additional output node to the flow, such as the MQOutput node. Wire the End of Data terminal on the FTEInput node to the In terminal of the MQOutput node, as shown in the following figure:



   The node connected to the End of Data terminal receives an empty message when the last record in the file is read.

5. Optional: To process the file based on details of the transfer, place a node such as the Route node after the FTEInput node.

   Details of the transfer are stored in the local environment, at `LocalEnvironment.FTE`.

6. Add the flow to a BAR file and deploy the BAR file.
7. Optional: Change the coordination queue manager.

Receiving a file using IBM Sterling Connect:Direct
Use the CDInput node to receive files from an IBM Sterling Connect:Direct network.

Before you begin

1. Read "IBM Sterling Connect:Direct overview and concepts" on page 953.
2. Ensure that IBM MQ is installed on the same machine as IBM App Connect Enterprise. Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the CDInput node.
3. Ensure that a queue manager has been specified to own the system queues that store information about the file transfer.
4. Create the system queues that will store the file transfer information. For more information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.

About this task

Multiple CDInput nodes can be deployed to the same integration server. Multiple CDInput nodes can read files transferred to the same directory without contention. Each file is processed only once, even if the nodes are deployed to separate integration servers.

On z/OS, when the CDInput node receives notification of the arrival of a dataset that it should process, the node copies that dataset into Unix System Services temporarily, before processing.

Complete the following steps to create a CDInput node message flow using the default settings. The task includes some optional steps for additional configuration, but detailed information about configuring the CDInput node is given on the property panels for the node, in the IBM App Connect Enterprise Toolkit. The default settings assume that you have a Connect:Direct server running on the same machine as IBM App Connect Enterprise, and that you are using the default ports.

Procedure

1. Drag a CDInput node onto a message flow and wire its Out terminal to an output node of your choice.
2. Optional: To process only a defined subset of files, configure the **Basic panel**.

   This action allows multiple CDInput nodes in the same integration server to receive specific files, depending on the directory or file filters specified.

   You can also specify whether, after processing, the file is left in its directory, renamed, or deleted.
3. To change how the node handles a message flow failure, configure the Retry panel.
4. Optional: To change how records are identified in the input file, configure the `Record detection` property on the **Records and Elements** panel.

   For example, you might want to specify that a record is fixed length, and set the record length.

   a) If you set the `Record detection` property to anything other than `Whole File`, drag an additional output node to the flow, such as the MQOutput node. Wire the End of Data terminal on the CDInput node to the In terminal of the MQOutput node. The node connected to the End of Data terminal receives an empty message when the last record in the file is read.
5. Optional: To process the file based on details of the transfer, place a node such as the Route node after the CDInput node.

   Details of the transfer are stored in the local environment, at `LocalEnvironment.CD`.
6. Set up the user name and password that are required for the CDInput node to connect to the primary Connect:Direct server by using the command.
7. Add the flow to a BAR file and deploy the BAR file.
8. Use IBM Sterling Connect:Direct to send a file to the local Connect:Direct server.

   The CDInput node processes this file.

Controlling how files are separated into records
Set the `Record detection` and other properties on node **Records and Elements** tabs to read files in different formats.

The following examples are based on the examples described in “Using a local file as input for your message flow” on page 975 and “Reading a file on a remote FTP or SFTP directory” on page 980. In each case, the input file to use, the property settings, and the expected results are described.

The examples, which describe using the FileInput node, can be applied to the FTEInput node, with the following provisos:

- The FTEInput node has no Basic tab.
- FTP and SFTP are not used to transmit files to an FTEInput node.

The examples can also be applied to the FileRead node, with the following provisos:

- The FileRead node propagates only one record from the file and, by default, this record is the first record in the file. The FileRead node can be configured to propagate a specific record. For more information see “Routing or enriching a message based on the contents of a file” on page 976.
- FTP and SFTP are not used to transmit files to an FileRead node.

Example 1. Records are separated by a DOS or UNIX line end

This example is identical to the one described in "Using a local file as input for your message flow" on page 975 or "Reading a file on a remote FTP or SFTP directory" on page 980. Create an input file called `test_input1.xml` with the following content:

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Each line ends with a line terminator.

The properties to set are:

Tab	Property	Value
Records and Elements	Record detection	Delimited
	Delimiter	DOS or UNIX Line End
	Delimiter type	Postfix

The FileInput node detects records that end with a DOS or UNIX line end and creates a message for each one that it finds.

The result is the propagation of three messages, as follows:

- Message 1:

  ```
  <Message>test1</Message>
  ```

- Message 2:

  ```
  <Message>testtwo</Message>
  ```

- Message 3:

  ```
  <Message>testthree</Message>
  ```

The DOS or UNIX line end is not part of any propagated message.

Example 2. Records are separated by a custom delimiter

Create an input file called `test_input2.xml` with the following content:

```
<Message>test01</Message>,<Message>test001</Message>,<Message>test0001</Message>
```

There must be no line terminator at the end of this file data; the XMLNSC parser ignores the line terminator if it is present.

In addition to the property settings described in "Using a local file as input for your message flow" on page 975 or "Reading a file on a remote FTP or SFTP directory" on page 980, set these properties:

Tab	Property	Value
Basic	File name or pattern	test_input2.xml
Records and Elements	Record detection	Delimited
	Delimiter	Custom Delimiter
	Custom delimiter	2C
	Delimiter type	Infix

The hexadecimal X'2C' represents a comma in ASCII. On other systems, you must use a different hexadecimal code.

The FileInput node detects the comma character and uses it to separate records. Because the value of the `Delimiter type` property is `Infix`, a comma is not required at the end of the file.

The result is the propagation of three messages, as follows:

- Message 1:

```
<Message>test01</Message>
```

- Message 2:

```
<Message>test001</Message>
```

- Message 3:

```
<Message>test0001</Message>
```

The comma character is not part of any propagated message. There are no commas in the bodies of the message in this example; if the message bodies did contain commas, the records would be split at those points resulting in incorrectly formed messages being propagated to the rest of the flow.

Example 3. Records are separated by a fixed number of bytes

Create an input file called `test_input3.xml` with the following content:

```
<Message>123456789</Message><Message>abcdefghi</Message><Message>rstuvwxyz</Message>
```

There must be no line terminator at the end of this file.

In addition to the property settings described in "Using a local file as input for your message flow" on page 975 or "Reading a file on a remote FTP or SFTP directory" on page 980, set these properties:

Tab	Property	Value
Basic	`File name or pattern`	`test_input3.xml`
Records and Elements	`Record detection`	`Fixed Length`
	`Length`	28

The FileInput node splits the input file into records each 28 bytes long.

The result is the propagation of three messages, as follows:

- Message 1:

```
<Message>123456789</Message>
```

- Message 2:

```
<Message>abcdefghi</Message>
```

- Message 3:

```
<Message>rstuvwxyz</Message>
```

Each message is 28 bytes long. If the file contains trailing bytes, for example a carriage return-line feed pair, a further message containing these bytes is propagated; the trailing bytes might or might not be recognized by the message domain, message model, and message type assigned to parse the message.

Example 4. Records are whole files

Create an input file called `test_input4.xml` with the following content:

```
<Message>Text string of a length decided by you, even including line
```

```
terminators, as long as it only contains this tag at the end.</Message>
```

There must be no line terminator at the end of this file; if there is one, it has no effect.

In addition to the property settings described in "Using a local file as input for your message flow" on page 975 or "Reading a file on a remote FTP or SFTP directory" on page 980, set these properties:

Tab	Property	Value
Basic	`File name or pattern`	`test_input4.xml`
Records and Elements	`Record detection`	`Whole File`

The FileInput node does not split the file; it propagates all of the content of the file as a single record to be parsed by the message domain, message model, and message type as specified on the node. In this example, you are using the XMLNSC parser and message set `xml1`, so the message is recognized.

The result is the propagation of one message, as follows:

- Message 1:

```
<Message>Text string of a length decided by you, even including line terminators, as long as
it only contains this tag at the end.</Message>
```

Trailing bytes (for example, line terminators) are included.

Example 5. Records are recognized as separate messages by the parser specified in the `Message domain` property

Create an input file called `test_input5.xml` with the following content:

```
<Message>Text string of a length decided by you  </Message><Message>and another</Message>
<Message>and another on a new line</Message>
```

Line terminators at the end of this file, or at the end of lines, are acceptable.

In addition to the property settings described in "Using a local file as input for your message flow" on page 975 or "Reading a file on a remote FTP or SFTP directory" on page 980, set these properties:

Table 24.

Tab	Property	Value
Basic	`File name or pattern`	`test_input5.xml`
Records and Elements	`Record detection`	`Parsed Record Sequence`

The FileInput node defers to the parser to determine the record boundaries. In this example, message set `xml1` in domain XMLNSC must recognize the complete `<Message>` XML format. XMLNSC absorbs trailing white space (for example, line terminators).

The result is the propagation of three messages, as follows:

- Message 1:

```
<Message>Text string of a length decided by you  </Message>
```

- Message 2:

```
<Message>and another</Message>
```

- Message 3:

```
<Message>and another on a new line</Message>
```

Trailing white space (for example, line terminators) are included in the messages.

Writing a file

Use the FileOutput, CDOutput, and FTEOutput nodes to write files.

About this task

This section contains the following topics:

Writing a file to your local file system

Use a FileOutput node to write a file to a specified directory on your local file system.

Before you begin

This example shows you how one combination of values in the `Record definition`, `Delimiter`, and `Delimiter type` properties result in the creation of a file from multiple messages. The example describes the FileOutput node of a message flow and assumes that the rest of the flow has been developed. It is also assumed that a Windows system is being used. To complete this example task, you must first have added a FileOutput node to a message flow. You must also ensure that the following messages are produced by the flow preceding the FileOutput node:

- Three input messages, which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```

  - Message 2:

    ```
    <Message>testtwo</Message>
    ```

  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```

- A final message, which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

  ```
  <thiscanbe>anything</thiscanbe>
  ```

Complete the following steps:

Procedure

1. Set the required node properties on the FileOutput node.

   The following table summarizes the FileOutput node properties that you must set, on which tab they appear, and the value that you must set in order to follow this example:

Tab	Property	Value
Basic	`Directory`	`C:\FileOutput\TestDir`
	`File name or pattern`	`test_output1.xml`

Tab	Property	Value
	Mode for writing to file	Stage in transit directory and move to output directory on Finish File
	Action if file exists	Time Stamp, Archive and Replace Existing File
	Replace duplicate archive files	Selected
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	FTP	Cleared

2. Deploy the message flow to the integration node. See Chapter 7, “Deploying integration solutions,” on page 2463.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.

Results

The following actions occur when you perform these steps:

1. The file is processed. In accordance with the values set in the properties of the FileOutput node, the node generates one record per message with a local file system line terminator after each one. The file contains the following data, each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

2. Records are accumulated in file `test_output1.xml` in the `C:\FileOutput\TestDir\mqsitransit` directory. When the final message is sent to the Finish File terminal, the file is moved to the output directory, `C:\FileOutput\TestDir` directory.
3. If a file of the same name exists in the output directory, the existing file is renamed and moved to the `mqsiarchive` directory. For example, the following file might be created:

   `C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml`

   If a file of this name exists in this archive directory, it is overwritten in accordance with the `Replace duplicate archive files` property selected on the FileOutput node.

What to do next

See “Setting the Record definition property for the FileOutput and FTEOutput nodes” on page 996 to see the results of running this task with different values set in the `Record definition`, `Delimiter`, and `Delimiter type` properties of the FileOutput node.

You can use the append mode of the FileOutput node to append to a file.

- Appending directly:

  An example is when an application uses the content of the file continually and any changes are seen as they occur. It might be that as soon as the records are seen the application loads the records into a database or a product catalog.

- Staging before appending:

  An example is when an application uses the content of the file during the day, but the application does not want any updates that occur during the day to be seen. IBM App Connect Enterprise batches the new record up in its transit directory and then, at a given time of the day (possibly outside normal working hours) the integration node appends to the file.

If a remote file exists, you can use the append mode of the FileOutput node to append to or replace an existing file. Data is either added to the end of the remote file or, if the file does not exist, the file is created.

Writing a file to a remote FTP or SFTP server
Use a FileOutput node to write a file to a directory on a remote FTP or SFTP server.

Before you begin

This example shows you how one combination of values in the `Record definition`, `Delimiter`, and `Delimiter type` properties result in the creation of a file from multiple messages. The example is an extension of the example described in "Writing a file to your local file system" on page 989, and describes the use of a FileOutput node in a message flow.

These instructions assume that you are using a Windows system, and that you have already created a message flow containing a FileOutput node. You also require the following resources:

- An FTP or SFTP server. Ensure that an FTP or SFTP server exists with the following settings, so that you can follow this example scenario:

  Server
  ftpserver.hursley.abc.com

  Port
  21 (for FTP) or *22* (for SFTP)

  Working directory
  /ftpfileoutput

  User ID
  myuserid

  Password
  mypassword

  These values are for the purposes of this example only. If you use other values, record them so that you can set the appropriate values when you follow the instructions in this task.

- A security identity. Use the **mqsisetdbparms** command to define a security identity called *myidentity* for your user and password details.

  If you want to connect to an FTP server, the security identity must have an `ftp::` prefix, to enable the file nodes to find the identity definition. For example:

  ```
  mqsisetdbparms -w workDir -n ftp::myidentity -u myuserid -p mypassword
  ```

  If you want to connect to an SFTP server, the security identity must have an `sftp::` prefix, as shown in the following example:

  ```
  mqsisetdbparms -w workDir -n sftp::myidentity -u myuserid -p mypassword
  ```

  You can also configure a connection to an SFTP server to use Public Key authentication, by specifying an SSH identity file and pass phrase, instead of a password. For example:

  ```
  mqsisetdbparms -w workDir -n sftp::myidentity -u myuserid -i identity_file -r passphrase
  ```

  To check what security credentials are already set, use the **mqsireportdbparms** command; see "Checking the password for a resource that is used by an integration server" on page 2630.

  For more information about configuring connections to an SFTP server, see "Transferring files securely by using SFTP" on page 998.
- The following messages, which must be produced by the message flow preceding the FileOutput node:
  - Three input messages. These messages are sent, in this order, to the In terminal of the FileOutput node:
    - Message 1:

      ```
      <Message>test1</Message>
      ```

    - Message 2:

      ```
      <Message>testtwo</Message>
      ```

    - Message 3:

      ```
      <Message>testthree</Message>
      ```

    These messages can be produced, for example, by the XMLNSC domain with a message set that recognizes XML, in the following form:

    ```
    <Message>...</Message>
    ```

  - A final message sent to the Finish File terminal of the FileOutput node after the first three messages have been sent:

    ```
    <thiscanbe>anything</thiscanbe>
    ```

Complete the following steps:

Procedure

1. Set the required node properties on the FileOutput node.

   The following table summarizes the FileOutput node properties that you must set, the tabs on which they are displayed, and the values that are used in this example:

Tab	Property	Value
Basic	`Directory`	`C:\FileOutput\TestDir`
	`File name or pattern`	`test_output1.xml`

Tab	Property	Value
	Mode for writing to file	Stage in transit directory and move to output directory on Finish File
	Action if file exists	Time Stamp, Archive and Replace Existing File
	Replace duplicate archive files	Selected
Records and Elements	Record definition	Record is Delimited Data
	Delimiter	Broker System Line End
	Delimiter type	Postfix
FTP	Remote transfer	Selected
	Transfer protocol	FTP or SFTP
	Remote server and port	ftpserver.hursley.abc.com
	Security identity	myidentity
	Server directory	/ftpfileoutput
	Transfer mode	ASCII (for FTP only)
	Action if remote file exists	Replace Existing File or Append to Existing File
	Retain local file after transfer	Selected

If you used other values for your FTP or SFTP server resource, use those values. The settings used here are identical to those settings used in the example in "Writing a file to your local file system" on page 989 except that the Remote transfer property has been selected and there are now properties on the **FTP** tab. If you clear the Remote transfer property, the node operates as it does in the example in "Writing a file to your local file system" on page 989; the properties on the **FTP** tab remain set but are ignored.

You can override the Remote server and port property on the node by setting a value in the local environment. For more information, see Local environment overrides for the remote server on the node.

2. Deploy the message flow to the integration server. See Chapter 7, "Deploying integration solutions," on page 2463.
3. Send the first three messages to the In terminal of the FileOutput node.
4. Send the final message to the Finish File terminal of the FileOutput node.
5. If the remote file exists, the mode for writing to remote file is used to specify if the file transferred replaces an existing file or appends data to the existing file. The transfer happens on the Finish File action. To support the use of Append you might have to update your FTP server configuration.

Results

The following actions occur when you perform these steps:

1. The file is processed. The FileOutput node generates one record per message with a local file system line terminator after each one. The file contains the following data, with each line terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system):

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

2. Records are accumulated in file `test_output1.xml` in the `C:\FileOutput\TestDir\mqsitransit` directory. When the final message is sent to the Finish File terminal, the file is moved to the remote FTP or SFTP server directory (because the `Remote transfer` property is selected). As a result, the file `/ftpfileoutput/test_output1.xml` is created.
3. If a file with the same name exists in the remote FTP or SFTP server directory, the existing file is overwritten.

   If the remote FTP server is not running on a Windows system and the `Transfer mode` property is set to `ASCII`, the character encoding and line terminator characters might be modified after transfer. For example, on a z/OS FTP server, the ASCII text is typically converted to EBCDIC, and the line terminator character pairs are replaced by EBCDIC new line characters (X'15'). Other FTP servers might treat ASCII transfers differently. If you are using SFTP, the `Transfer mode` property is ignored and files are sent as Binary files.
4. Because the `Retain local file after transfer` property is selected, the local file is not deleted but is moved from the `mqsitransit` subdirectory to the output directory, `C:\FileOutput\TestDir`. If a file with the same name exists in the output directory, the existing file is renamed and moved to the `mqsiarchive` directory. For example, the following file might be created:

   `C:\FileOutput\TestDir\mqsiarchive\20081124_155346_312030_test_output1.xml`

   However, if a file with this name exists in this archive directory, it is overwritten according to the value of the `Replace duplicate archive files` property set on the FileOutput node.

What to do next

For more information, see "Setting the Record definition property for the FileOutput and FTEOutput nodes" on page 996, which shows the results of running this task with different values set in the `Record definition`, `Delimiter`, and `Delimiter type` properties of the FileOutput node.

Sending a file by IBM MQ File Transfer Edition

Send files to an existing IBM MQ File Transfer Edition network.

Before you begin

1. Read about IBM MQ File Transfer Edition.
2. Get the following information from your IBM MQ File Transfer Edition administrator:
   - The name of the remote IBM MQ File Transfer Edition agent to which the file is to be sent.
   - The name of the destination queue manager.
   - The name of the output file.
3. Ensure that IBM MQ is installed on the same machine as IBM App Connect Enterprise. Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the FTEOutput node.
4. Ensure that a queue manager has been specified to own the system queues that store information about the file transfer.
5. Create the system queues that will store the file transfer information. For more information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.
6. Create a message flow that contains an input node.

About this task

Detailed information about configuring the FTEOutput node is given on the property panels for the node, in the IBM App Connect Enterprise Toolkit.

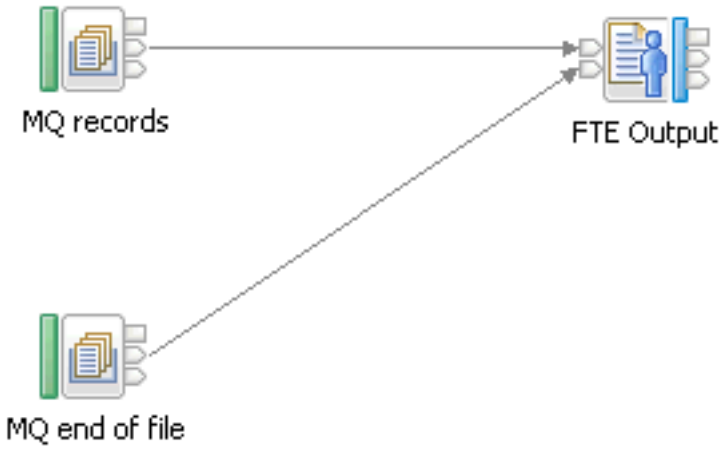
Procedure

1. Drag an FTEOutput node onto the message flow, and wire its In terminal to the input node.
2. On the **Basic** panel, set values for the `Destination agent` and `Destination file name` properties.

   Configuring just these two properties is enough if you want to send all the input message tree as a single record in the output file.
3. Optional: On the **Basic** panel, set a value for the `Destination queue manager` property.
4. Optional: To specify a location in the input message tree for the data to be sent, configure the `Data location` property on the **"Request properties"** panel.
5. Optional: To change how records are placed in the output file, configure the `Record definition` property on the **"Record definition"** panel.

   For example, you might want to specify that a record is fixed length, and set the record length.

   a) If you set the `Record definition` property to anything other than `Record is Whole File`, drag a node such as the MQOutput node to the flow, and wire its Out terminal to the Finish File terminal on the FTEOutput node, as shown in the following figure:



   The node connected to the Finish File terminal must have logic to determine the last record in the file.
6. Optional: To set properties for the transfer dynamically, place a node such as the Compute node or Mapping node before the FTEOutput node.

   You can override the following properties:

   - `Destination agent`
   - `Destination file directory`
   - `Destination file name`
   - `Destination queue manager`
   - `Job name`
   - `Overwrite files on destination`

   Configure the node to write the overrides to the `LocalEnvironment.Destination.FTE` subtree.
7. Add the flow to a BAR file and deploy the BAR file.

Setting the `Record definition` property for the FileOutput and FTEOutput nodes
Set the properties on the**Records and Elements** tab of the node to write files in different formats.

The following examples are based on those described in "Writing a file to your local file system" on page 989 and "Writing a file to a remote FTP or SFTP server" on page 991. The FTEOutput does not archive, and files are transferred using IBM MQ File Transfer Edition, rather than FTP or SFTP. In all examples, it is assumed that the same messages are sent to the FileOutput node; three to the In terminal and one to the Finish File terminal:

- Three input messages which are sent, in this order, to the In terminal of the FileOutput node:
  - Message 1:

    ```
    <Message>test1</Message>
    ```

  - Message 2:

    ```
    <Message>testtwo</Message>
    ```

  - Message 3:

    ```
    <Message>testthree</Message>
    ```

  These messages can be produced, for example, by the XMLNSC domain with a message set which recognizes XML with the following form:

  ```
  <Message>...</Message>
  ```

- A final message which is sent to the Finish File terminal of the FileOutput node after the first three messages have been sent. It does not matter what this message contains.

The following examples describe the contents of the file or files produced; the disposition of the files created is as in the "Writing a file to your local file system" on page 989 and "Writing a file to a remote FTP or SFTP server" on page 991 topics.

Example 1. Records written are separated by a DOS or UNIX line end

This example is identical to the one described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991. Specify the node's properties as described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991.

These properties result in one file being written. The file contains three records each terminated by a local system line terminator. On a Windows system, this is a carriage return (X'0D') line feed (X'0A') pair of characters; on UNIX systems it is X'0A'.

```
<Message>test1</Message>
<Message>testtwo</Message>
<Message>testthree</Message>
```

Example 2. Records written are separated by a custom delimiter

In addition to the property settings described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991, set these properties on the **Records and Elements** tab:

Property	**Value**
`Record definition`	`Record is Delimited Data`
`Delimiter`	`Custom Delimiter`
`Custom delimiter`	`0D0A`

Property	Value
Delimiter type	Postfix

The hexadecimal X'0D0A' represents a carriage return character followed by a line feed character. On a Windows system, this results in a file identical to the one created in Example 1. On other systems, the result might differ from the result in Example 1; Example 1 uses local system line end characters, whereas Example 2 always puts the X'0D0A' sequence at the end of each line.

Example 3. Records written are padded to a fixed length

In addition to the property settings described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991, set these properties on the **Records and Elements** tab:

Property	Value
Record definition	Record is Fixed Length Data
Length (bytes)	30
Padding bytes (hexadecimal)	2A

The hexadecimal character X'2A' represents an asterisk character in ASCII.

The length of each incoming message is 24 bytes, 26 bytes, and 28 bytes respectively. The required fixed length of each record is 30 bytes. Each record is therefore padded by an extra 6 bytes, 4 bytes, and 2 bytes respectively, using the hexadecimal character X'2A'.

One file is written. It contains a single line:

```
<Message>test1</Message>******<Message>testtwo</Message>****<Message>testthree</Message>**
```

Example 4. Records written are not separated by delimiters or padding

In addition to the property settings described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Unmodified Data

The records are concatenated with no padding or delimiters.

One file is written with the following content:

```
<Message>test1</Message><Message>testtwo</Message><Message>testthree</Message>
```

There are no trailing bytes or line terminators.

Example 5. Records are written as whole files

In addition to the property settings described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991, set this property on the **Records and Elements** tab:

Property	Value
Record definition	Record is Whole File

Three files are created, each containing one record:

- File 1:

```
<Message>test1</Message>
```

- File 2:

```
<Message>testtwo</Message>
```

- File 3:

```
<Message>testthree</Message>
```

Each of these files is created with the same name, one by one, in the mqsitransit directory. If you are following the example in "Writing a file to a remote FTP or SFTP server" on page 991, each file is transferred to the remote FTP server. However, because each file overwrites the previous one, only the third file remains when the task is complete.

After optional transfer, if a copy is retained, each file is moved to the output directory, `C:\FileOutput\TestDir`. In accordance with the properties on the FileOutput node as described in "Writing a file to your local file system" on page 989 or "Writing a file to a remote FTP or SFTP server" on page 991, the second file moved displaces the first file from the output directory which is moved to the mqsiarchive subdirectory with a time stamp added to the file name. When the third file is moved to the output directory, it displaces the second file, causing it to be moved to the mqsiarchive subdirectory and renamed. The final result is files similar to these:

```
C:\FileOutput\TestDir\mqsiarchive\20071101_165346_312030_test_output1.xml
C:\FileOutput\TestDir\mqsiarchive\20071101_165347_312030_test_output1.xml
C:\FileOutput\TestDir\test_output1.xml
```

being File 1, File 2, and File 3 respectively. If FTP processing was enabled, File 3 would also be in the remote FTP server directory and called `test_output1.xml`.

Transferring files securely by using SFTP

You can transfer files securely by using the Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

Use the properties on the FTP tab of the FileInput and FileOutput nodes to configure the secure transfer of files.

You can also use the FTP Server policy to configure other SFTP properties, including the cipher to be used for SFTP communication, compression level, and strict known host checking (see FTP Server policy (FtpServer)).

See the following topics for more information about configuring secure file transfer:

- "Configuring SFTP file transfer" on page 998
- "Known host checking" on page 999

Configuring SFTP file transfer

Use an FTP Server policy to specify the Secure File Transfer Protocol (SFTP) settings for a message flow, and to override the SFTP settings that are specified on the FileInput and FileOutput nodes.

About this task

The settings that you specify by using an FTP Server policy are read and validated when the message flow starts, and are used to configure any SFTP connections that are made for the node. The policy can override any or all of the remote transfer properties on the **FTP** tab of the FileInput and FileOutput nodes. For more information about the settings that you can specify with an FTP Server policy, see FTP Server policy (FtpServer).

You can configure strict host key checking and specify your own known hosts file, or you can turn off strict host key checking and use the known hosts files that are created and managed by the integration node.

Multiple policies can specify the same host and port, even with different known hosts files. FTP defaults to port 21 and SFTP defaults to port 22, which is the SSH default port. If you set the port and specify an FTP connection to an SFTP server (or specify an SFTP connection to an FTP server) a connection error occurs and a message is added to the event log.

You can use the FTP Server policy to configure the following SFTP settings:

- Cipher used for SSH/SFTP communication
- Compression level
- Strict known host checking
- Protocol (FTP/SFTP) for nodes to use for remote file transfer
- Location of a known hosts file when strict known host checking is set to **Yes**

Procedure

1. Create an FTP Server policy with the required parameter values by using the Policy editor (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. In the FileInput and FileOutput nodes, specify the name of the FTP Server policy in the **Remote server and port** property on the **FTP** tab.

Known host checking
Use known host checking to control which hosts the integration node can connect to, and to verify the identity of those hosts.

Known host checking enables the integration node to protect the messages in the message flow from unauthorized attempts to intercept the data (sometimes known as *man-in-the-middle attacks*). Known hosts files contain the SSH keys of the hosts to which the integration node can connect. On z/OS systems, known hosts files and SSH identity files are stored in EBCDIC format, and on other operating systems they are stored in ASCII format.

Before it connects to a host to receive or transfer a file (using the FileInput or FileOutput nodes), the integration node checks the host key against the keys that are stored in the known hosts file. If the host key does not match an existing entry for the host in the known hosts file, the connection fails. If the host is new (and has no entry in the known hosts file), the result depends on whether strict known host checking is turned on or off.

You can specify whether strict known host checking is turned on or off by setting the **Strict host key checking** property of the FTP Server policy (see FTP Server policy (FtpServer)).

Strict known host checking

When strict known host checking is turned on (by setting the **Strict host key checking** property to Yes), the integration server connects only to known hosts with valid SSH host keys that are stored in the known hosts file. If you use strict known host checking, you must create your own known hosts file, in OpenSSL PEM format, containing the SSH keys of your trusted hosts. When the integration server attempts to make a connection to a host, it checks the host key against the contents of the known hosts file. If the key is not in the known hosts file, the connection fails and a BIP3371 error occurs.

You can have multiple known hosts files and specify a different one for each policy. The known hosts files that you provide for strict known host checking are not modified by the integration server.

Non-strict known host checking

When strict known host checking is turned off (by setting the **Strict host key checking** property to No) the integration server connects only to known hosts with valid keys or to new hosts to which it has not connected before. If the integration server attempts to connect to a new host (to which it has not connected previously), the integration server makes the connection, accepts the host's SSH key, and stores it in the known hosts file. When the integration server tries to connect to the same host on subsequent occasions, the host key is checked against the key stored in the known hosts file, and, if the

key matches, the connection is made. However, if the host key is different from the one stored in the known hosts file, the connection attempt fails and a BIP3371 error occurs.

When strict known host checking is turned off, the known hosts file is managed by the integration server. One known hosts file exists for each integration server.

BIP3371 error message

The BIP3371 error message might indicate that there has been an unauthorized attempt to intercept a message. However, the connection failure (and resulting BIP3371 error message) might be caused by a change to the host's SSH key at some time following its first connection to the integration server. For example, if the SSH server is reinstalled, it is assigned a new key, which is not found in the known hosts file and is therefore not accepted by the integration server. As a result, the connection fails and the BIP3371 error message is shown.

If you know that the SSH host key has changed (as a result of a recent SSH server reinstallation, for example), you can solve the connection failure by modifying the known hosts file:

1. Stop the message flow.
2. Edit the known hosts file.

   If you have strict known host checking turned off, remove the host's entry from the known hosts file that is managed by the integration server. The known hosts file is in the `\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts` subdirectory of the directory in which IBM App Connect Enterprise is installed. For example, on Windows the default directory is `C:\ProgramData\IBM\MQSI\components\`*`BROKERNAME-NAME`*`\`*`EG-UID`*`\config\known_hosts`. On UNIX systems the default directory is `/var/mqsi/components/`*`BROKERNAME-NAME`*`/`*`EG-UID`*`/config/known_hosts`.

   When the integration server attempts to reconnect, it adds the new host key to the known hosts file.
3. Restart the message flow.

Managed file transfers using WebSphere MQ File Transfer Edition

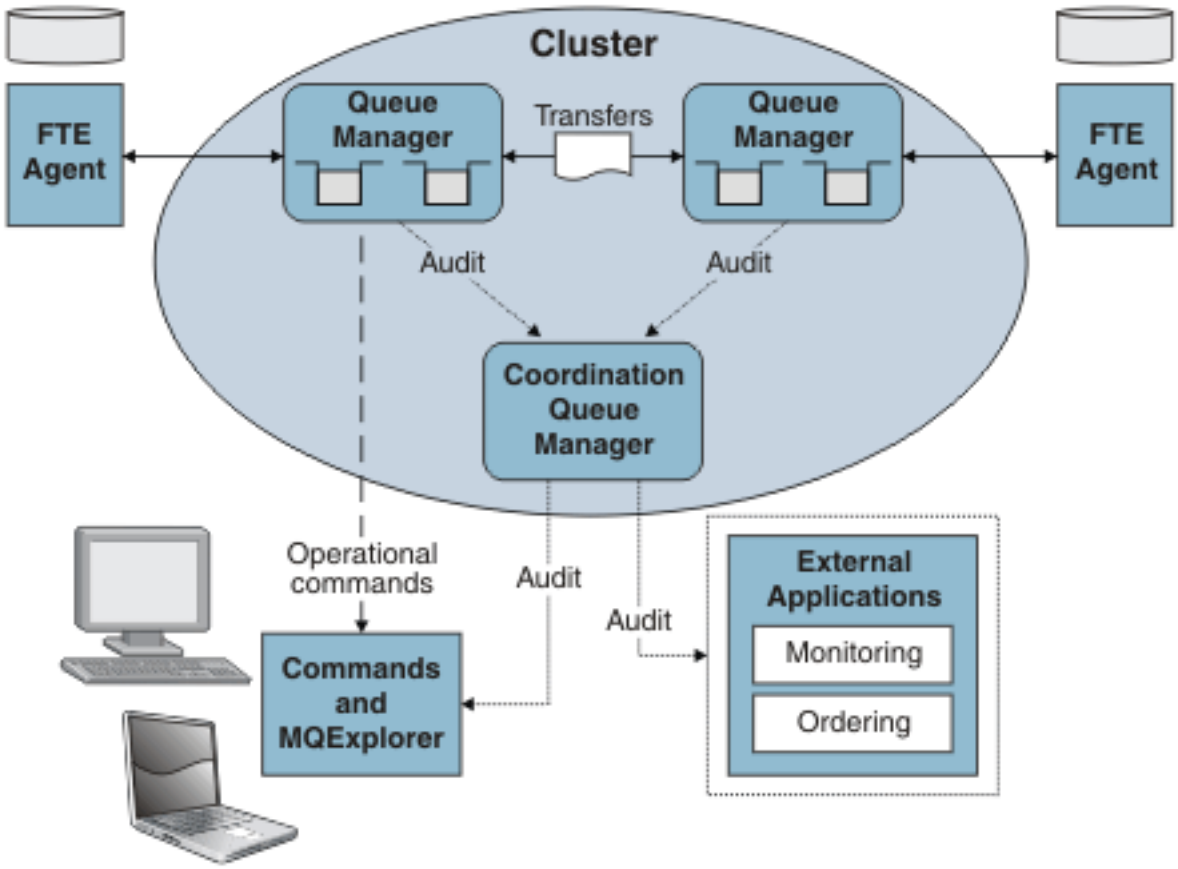
Transfer files, with file transfer metadata, in a timely and reliable manner.

- Transfer metadata is supported, providing more flexible file processing.
- Transfers are timely.
- Resource statistics are available.

About IBM MQ File Transfer Edition

WebSphere MQ File Transfer Edition V7.0 delivers a reliable managed file transfer solution for moving files between IT systems without the need for programming. Files that are larger than the maximum individual WebSphere MQ message size can be moved. A log of file movements demonstrates that business data in files is transferred with integrity from source file system to destination file system.

Using the WebSphere MQ File Transfer Edition nodes offers seamless integration with your existing IBM MQ File Transfer Edition network. The following diagram shows a typical IBM MQ File Transfer Edition network:



The main components and concepts of IBM MQ File Transfer Edition are:

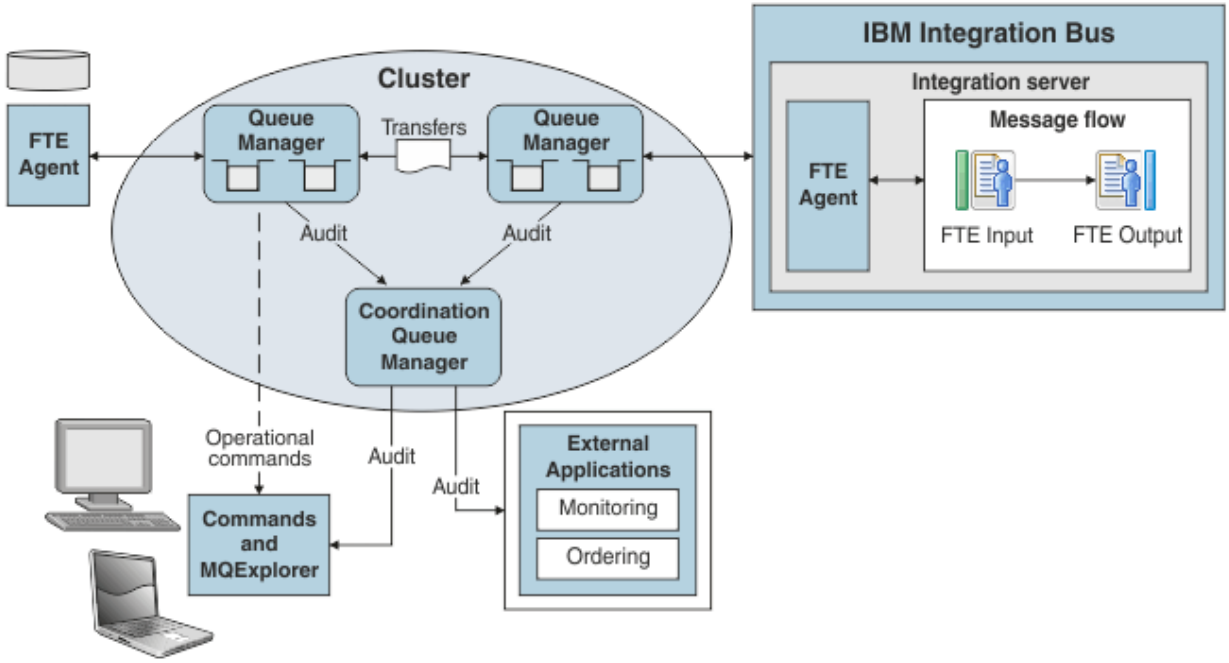
agent
A process that forms the end point of a transfer (source or destination). An agent is an IBM MQ application, and is connected to one queue manager. Many agents can be connected to the same queue manager. Operational commands can be sent to an agent (for example, to request a transfer) by putting an XML message on a particular queue on the agent's queue manager.

coordination queue manager
One queue manager in the topology takes on the responsibility of the coordination queue manager. All agents register with the coordination queue manager and also send audit information about each transfer. The coordination queue manager is responsible for publishing that audit information to external monitoring applications.

transfer
A transfer is a movement of one or more files from one agent to another. The transfer goes direct from one agent's queue manager to the other agent's queue manager, not via the coordination queue manager. The transfer takes place even if the coordination queue manager is not running.

transfer log
The IBM MQ Explorer plug-in tool includes a Transfer Log view that subscribes to the coordination queue manager for the audit information. The view displays information about every transfer that occurs in a given topology.

How IBM App Connect Enterprise fits in



agent

An agent runs in each integration server that has deployed flows containing WebSphere MQ File Transfer Edition nodes. The agent is responsible for receiving and initiating all IBM MQ File Transfer Edition transfers.

You do not need to start or stop this agent; if a flow that contains WebSphere MQ File Transfer Edition nodes is deployed, the agent is running. The agent is stopped only when the integration server is stopped.

transfer

Multiple FTEOutput nodes can be deployed to the same integration server. FTEOutput nodes can send one file per transfer. Each file can have multiple records. Each record can have multiple elements. Transfers from the FTEOutput are non-blocking; an error occurs if another transfer is outstanding with the same file name.

Multiple FTEInput nodes can be deployed to the same integration server. Multiple FTEInput nodes can read files transferred to the same directory without contention. Each file is processed only once, even if the nodes are deployed to separate integration servers.

When sending a file, you can dynamically set the following properties:

- `Destination agent`
- `Destination file directory`
- `Destination file name`
- `Destination queue manager`
- `Job name`
- `Overwrite files on destination`

Put a node such as the Compute node or Mapping node before the FTEOutput node.

When receiving files, you can apply filters. If an integration server has more than one FTEInput node, each node receives only the appropriate files. You can also determine what happens after the file has been processed (the file is left in its existing destination directory, left with a timestamp added, or deleted). See the Basic tab on the node for details.

You can combine the FTEInput and FTEOutput nodes to create a request and reply flow. See the sample for details.

Using elements in the local environment, you can call a program on the destination agent before starting the transfer, or when the transfer is finished. See "LocalEnvironment.Destination.FTE fields" on page 968 for details.

Overview of using WebSphere MQ File Transfer Edition nodes

You do not need to configure the IBM MQ File Transfer Edition code that runs in IBM App Connect Enterprise.The following nodes are provided:

FTEOutput
FTEInput

Take the following steps to use the nodes to send or receive data across an existing IBM MQ File Transfer Edition network:

1. Ensure that IBM MQ is installed on the same machine as IBM App Connect Enterprise. Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the FTEInput and FTEOutput nodes.
2. Ensure that a queue manager has been specified to own the system queues that store information about the file transfer.
3. Create the system queues that will store the file transfer information. For more information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.
4. Create a flow that includes one of the IBM MQ File Transfer Edition nodes.
5. Configure the node.
6. Deploy the flow.

Monitoring the license usage of WebSphere MQ File Transfer Edition in IBM App Connect Enterprise

A message is written to the system log for every integration server that has a IBM MQ File Transfer Edition node deployed.

Initiating a managed file transfer using IBM Sterling Connect:Direct

Use a CDOutput node to send a file from a specified directory on your primary Connect:Direct server (PNODE) to a filename and directory on a secondary Connect:Direct server (SNODE).

Before you begin

1. Ensure that IBM MQ is installed on the same machine as IBM App Connect Enterprise. Information about file transfers is held on storage queues that are controlled by IBM MQ, so you must install IBM MQ on the same computer as IBM App Connect Enterprise if you want to use the capabilities that are provided by the CDOutput node.
2. Ensure that a queue manager has been specified to own the system queues that store information about the file transfer.
3. Create the system queues that will store the file transfer information. For more information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.
4. Create a message flow containing a CDOutput node; for example, create a simple flow containing an MQInput connected to a CDOutput node.

About this task

Multiple CDOutput nodes can be deployed to the same integration server. CDOutput nodes can send one file per transfer. Each file can have multiple records; each record can have multiple elements. Transfers from the CDOutput node are non-blocking.

Procedure

1. Set the required node properties on the CDOutput node.

   If you set the `Destination file name` only, and leave all the other options at their default values, the file is transferred:

   - From the primary Connect:Direct server (PNODE) back to itself
   - Into the default transfer directory using the default process name.

   Additionally, the file is created if it does not exist, or replaced if it already exists.

   The following table summarizes the CDOutput node properties that you can set, on which tab they appear, and a value that you can select:

Tab	Property	Value
Basic	`Process name`	You can use any name you want for the process. Note, however, that the name must be a maximum of eight characters and cannot contain any spaces.
	`SNODE`	`The secondary` Connect:Direct server to which the file is being transferred.
	`Destination file directory`	`TestDir` on secondary Connect:Direct server (SNODE).
	`Destination file name`	`Filename` on secondary Connect:Direct server (SNODE).
	`Disposition`	`RPL`
	`Transfer mode`	`Text mode`

2. Setup the *username* and *password* that is required for the CDOutput node to connect to the primary Connect:Direct server using the command.
3. Deploy the message flow (see Chapter 7, “Deploying integration solutions,” on page 2463).
4. Send the file to the In terminal of the CDOutput node.

Results

The following actions occur when you perform these steps:

1. The file is constructed in accordance with the values set in the properties of the CDOutput node.
2. The file is staged in the local file system and then a command is sent to the Connect:Direct server to cause the transfer to occur.
3. If a file of the same name exists in the selected directory on the secondary Connect:Direct server, the processing of the existing file is determined by the value of the *Disposition* property; in this example, the file is replaced.

   Once the transfer has completed the local, staged file is deleted.

Note, that when sending a file, you can dynamically set the following properties:

- `Secondary Connect:Direct server (SNODE)`

- Process name
- Accounting data
- Destination file directory
- Destination file name
- Copy from options
- Copy to options

You have complete control of the Copy statements.

For example:

```
LocalEnvironment.Destination.CD.Copy.To.Option.PERMISS = '777'
```

causes IBM Sterling Connect:Direct to set the permissions on the destination file to 777 (or RWX RWX RWX) if the destination file is on a UNIX operating system, or within Unix System Services on z/OS.

However, if you enter an incorrect format, a syntax error is detected when the process script is submitted and this causes an error to be thrown in the node.

For example:

```
LocalEnvironment.Destination.CD.Copy.To.Option.PERMISS = '7xddd'
```

causes a syntax error because '7xddd' is not of the format nnn. When an error occurs the process script is available in the exception thrown and the user trace.

Tip: To help you with problem determination, you can view the script generated by the CDOutput node by turning on user trace.

WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository (WSRR) is a central repository of entities. A wide range of entities can be stored and retrieved, including user-defined concepts and definitions related specifically to web services, such as WSDL services, service interfaces, and associated policies.

You can configure a message flow to dynamically retrieve resources from WSRR at run time, and to use and expose those resources in the message flow. You can therefore defer the decision about which resources you want to use until run time, rather than deciding at deployment time.

WSRR has specific support for many of the document types that are associated with web services, including generic XML documents, WSDL, and SCDL. For example, when you load a WSDL document into WSRR it also identifies and stores its individual logical components, such as the service and port type.

Use the WSRR nodes (the RegistryLookup and EndpointLookup nodes) to create message flows that retrieve data dynamically from WSRR. Data is retrieved according to search criteria defined by node properties, possibly supplemented or overridden by local environment definitions. The retrieved data is placed in the local environment tree, which makes the data available to subsequent nodes. The input message that is received by the node is propagated to the output terminal unchanged.

Use the RegistryLookup node to submit generic queries to WSRR. Entities that are returned by the query are stored in the ServiceRegistry output tree in the local environment. You can also specify that details of the relationships between the returned entities and other entities that they reference are represented in the ServiceRegistry output tree.

Use the EndpointLookup node to submit queries for web service endpoints. This node is tailored to retrieve WSDL port definitions that implement a specified WSDL portType. The details of service endpoints that match the specified criteria are placed in the ServiceRegistry output tree in the local environment. If the node is configured to return a single matching service endpoint, the web service URL destination that is used by the SOAP and HTTP request nodes is also overridden in the local environment. If the node is configured to return all matching service endpoints, the local environment is not set up automatically for the SOAP and HTTP request nodes. In this case, the local environment tree might contain data for multiple service endpoints, and the message flow interprets and uses this information.

Set the configuration parameters for the WSRR nodes to specify how IBM App Connect Enterprise interfaces with your WSRR server.

- Use the **connectionTimeout** parameter to set a system-wide connection timeout for queries that are issued by the EndpointLookup and RegistryLookup nodes. If a query result is not returned from the WSRR server before the connection timeout period expires, the configured error handling is invoked.
- Use the **needCache** parameter to enable the IBM App Connect Enterprise WSRR cache. The cache is used to store results from queries that are issued by the EndpointLookup and RegistryLookup nodes.
- If the IBM App Connect Enterprise WSRR cache is enabled, use the **timeout** parameter to set a system-wide cache timeout. The cache timeout controls how long results from queries that are stored in the cache are used before the query is reissued.

For information about the specific levels of WSRR that are supported by IBM App Connect Enterprise, see IBM App Connect Enterprise system requirements.

The following topics provide further information about working with WSRR:

Configuration parameters for the WebSphere Service Registry and Repository nodes

These parameters affect the integration node configuration when interfacing with WebSphere Service Registry and Repository (WSRR).

To set these parameters, you must edit the `WSRRConnectorProvider` entries in the `ConnectorProviders` section of the `node.conf.yaml` or `server.conf.yaml` configuration file. When you create a new integration node or a new independent integration server, the `.conf.yaml` configuration file contains the following subset of the WSRR parameters. All the other WSRR parameters have default settings which generally do not need to be changed. If you need to change any of the other WSRR parameters, you must add them to the `.conf.yaml` configuration file.

```
ConnectorProviders:
    #WSRRConnectorProvider:    # Requires the optional WSRR component install
    #endpointAddress: 'https://host:9443/WSRR8_0/services/WSRRCoreSDOPort'   # WSRR server
 endpoint url
    #needCache: true                                                        # enable WSRR cache
    #predefinedCacheQueries: ''                                             # semicolon-
separated XPath querys to initialize WSRR cache at start-up
    #enableCacheNotification: false                                         # enable WSRR cache
 notification
    #locationJNDIBinding:  'iiop://host:2809'                               # WSRR cache WAS
 JMS provider JNDI bindings url
```

The following table describes the parameters that are used in the `node.conf.yaml` or `server.conf.yaml` configuration file to configure the WebSphere Service Registry and Repository (WSRR).

Configuration Setting Name	Default value	Description
endpointAddress	For all versions of WSRR: `http://hostname:9080/WSRRCoreSDO/services/WSRRCoreSDOPort`	Endpoint of the WSRR server. Where *hostname:port* is the domain qualified hostname and port of your WSRR server. To connect to a secure WSRR, specify `https://` instead of `http://`. For more information, see "Accessing a secure WebSphere Service Registry and Repository repository" on page 1011. For more information about the specific levels of WSRR that are supported by IBM App Connect Enterprise, see IBM App Connect Enterprise system requirements.
connectionTimeout	180	The WSRR connection timeout period in seconds. The default value is 180 seconds. Set a value that is sufficiently long for complex queries to complete.

For more information about WSRR endpoint addresses, see WSRR web service interface URLs.

The following table describes the parameters for Cache that are configured by updating the `node.conf.yaml` or `server.conf.yaml` configuration file for the relevant integration node or server.

Configuration Setting Name	Default value	Description
needCache	true	Enable IBM App Connect Enterprise WSRR cache.
timeout	100000000	The timeout value for the cache. The cache expiry time in milliseconds. (The default value of 100000000 milliseconds is approximately 27.8 hours).
predefinedCacheQueries	None	A list of semicolon-separated WSRR XPath query expressions with which to initialize the IBM App Connect Enterprise WSRR cache at startup. These WSRR XPath query expressions are defined by the WSRR SOAP interface query expression language, and can include an optional depth specifier extension.

The following table describes the parameters for Cache Notification that are configured by updating the `node.conf.yaml` or `server.conf.yaml` configuration file for the relevant integration node or server.

Configuration Setting Name	Default value	Description
enableCacheNotification	false	Enable IBM App Connect Enterprise WSRR Cache Notification.
refreshQueriesAfterNotification	true	When a notification is received from WSRR, if `refreshQueriesAfterNotification` is set to `true`, the cache is updated with the new version of the object immediately; if `false`, the cache is updated on the next request.

Configuration Setting Name	Default value	Description
connectionFactoryName	jms/SRConnectionFactory	The name of the WSRR WebSphere Application Server JMS provider JMS connection factory for Cache Notification.
initialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory	The name of the WSRR WebSphere Application Server JMS provider JMS context factory for Cache Notification.
locationJNDIBinding	iiop://*hostname*:2809/	The URL to the WebSphere Application Server JMS provider JNDI bindings, where *hostname* is variable.
secureJNDIPropertiesFiles	C:\SSLPropsFiles	The directory where your sas.client.props and the ssl.client.props files are stored when using secure IIOP for WSRR cache notification.
subscriptionTopic	jms/SuccessTopic	The topic name used to receive WebSphere Application Server JMS provider Cache Notification.

For more information about accessing a secure WSRR server, see "Accessing a secure WebSphere Service Registry and Repository repository" on page 1011.

Changing the configuration parameters for the WebSphere Service Registry and Repository nodes

To change the configuration parameters for the WebSphere Service Registry and Repository nodes, edit the `.conf.yaml` configuration file for the integration node, or integration server.

About this task

Any properties that you set for an integration node in the `node.conf.yaml` file, are inherited by the integration servers that it owns. However, you can change any of the properties of an integration server that is owned by an integration node by modifying them in the appropriate `server.conf.yaml` file. You can change any of the properties of an independent integration server by modifying them in the appropriate `server.conf.yaml` file. For more information about configuring integration servers, see Configuring an integration server by modifying the server.conf.yaml file.

To set the WSRR parameters, you must edit the `WSRRConnectorProvider` entries in the `ConnectorProviders` section of the `node.conf.yaml` or `server.conf.yaml` configuration file. When you migrate your IBM Integration Bus 10.0 integration servers and integration nodes to IBM App Connect Enterprise 12.0 by using the **mqsiextractcomponents** command, the values of the DefaultWSRR configurable service properties are placed in the `ConnectorProviders` section of the `.conf.yaml` file. When you create a new integration node or a new independent integration server, the `.conf.yaml` configuration file contains the following subset of the WSRR parameters. All the other WSRR parameters have default settings, which generally do not need to be changed. If you need to change any of the other WSRR parameters, you must add them to the `.conf.yaml` configuration file.

```
ConnectorProviders:
    #WSRRConnectorProvider:    # Requires the optional WSRR component install
    #endpointAddress: 'https://host:9443/WSRR8_0/services/WSRRCoreSDOPort'   # WSRR server
 endpoint url
    #needCache: true                                                        # enable WSRR cache
    #predefinedCacheQueries: ''                                             # semicolon-
separated XPath querys to initialize WSRR cache at start-up
```

```
   #enableCacheNotification: false                                    # enable WSRR cache
notification
   #locationJNDIBinding:  'iiop://host:2809'                          # WSRR cache WAS
JMS provider JNDI bindings url
```

For more information about configuration parameters that affect WSRR use, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1006.

Complete the following steps to update the configuration parameters for WSRR:

Procedure

1. Ensure that the integration node or independent integration server is running.

   For more information, see "Starting and stopping an integration node" on page 247 and "Starting an integration server" on page 250.
2. Edit the *connectionFactoryName* parameter in the `.conf.yaml` file to show the name of the WSRR WebSphere Application Server WebSphere® Application Server JMS provider JMS connection factory for Cache Notifications. The default value is `'jms/SRConnectionFactory.connectionFactoryName'`

   ```
   connectionFactoryName: 'jms/SRConnectionFactory' # The name of the WSRR Application Server
    JMS provider JMS connection factory for Cache Notifications. The default value is 'jms/
   SRConnectionFactory'.
   ```
3. Edit the *connectionTimeout* parameter in the `.conf.yaml` file to show the WSRR connection timeout period in seconds. The default value is 180 seconds (3 minutes).

   ```
   connectionTimeout: 180 # The WSRR connection timeout period in seconds. The default value is
    180 seconds (3 minutes).
   ```
4. Set the *enableCacheNotification* parameter in the `.conf.yaml` file to `true` to enable IBM App Connect Enterprise WSRR Cache Notification. The default value is false.

   ```
   enableCacheNotification: true # The default value is false. Select true to enable IBM® App
    Connect Enterprise WSRR Cache Notification.
   ```
5. Edit the *endpointAddress* parameter in the `.conf.yaml` file to show the location or endpoint of the WSRR server.

   ```
   endpointAddress: 'http://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort'
      #The location or endpoint of the WSRR server. The default value for all versions of WSRR
    is 'http://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort'
   ```

   The default value for all versions of WSRR is `'http://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort'`

   For example, to connect to a server that uses WSRR v7.5 compatibility use the following values:

   ```
   'http://fill.in.your.host.here:9080/WSRR7_5/services/WSRRCoreSDOPort'
   ```

   For more information about WSRR endpoint addresses, see WSRR web service interface URLs.

   For more information about the specific levels of WSRR that are supported by IBM App Connect Enterprise, see IBM App Connect Enterprise system requirements web page.
6. Edit the *initialContextFactory* parameter in the `.conf.yaml` file to show the name of the WSRR WebSphere Application Server WebSphere® Application Server JMS provider JMS connection factory for Cache Notifications. The default value is `'com.ibm.websphere.naming.WsnIitialContextFactoryCopy'`

   ```
   initialContextFactory: 'com.ibm.websphere.naming.WsnIitialContextFactoryCopy' # The name
    of the WSRR WebSphere Application Server JMS provider JMS context factory for Cache
    Notifications.

     The default value is 'com.ibm.websphere.naming.WsnIitialContextFactoryCopy'
   ```

7. Edit the *locationJNDIBinding* parameter in the `.conf.yaml` file to show the URL to the WebSphere Application Server JMS provider JNDI bindings. The default value is `'iiop://host_name:2809/'`

```
locationJNDIBinding: 'iiop://host_name:2809/' # The URL to the WebSphere Application Server
 JMS provider JNDI bindings. The default value is 'iiop://host_name:2809/'
```

8. Edit the *secureJNDIPropertiesFiles* parameter in the `.conf.yaml` file to show the directory where your.props files are stored. For example, `'C:\SSLPropsFiles'`

```
secureJNDIPropertiesFiles: 'C:\SSLPropsFiles' # The directory where your .props files are
 stored. For example 'C:\SSLPropsFiles'
```

9. Set the *needCache* parameter in the `.conf.yaml` file to `true` to indicate that the IBM App Connect Enterprise cache is enabled. The default value is `true`

```
needCache: true # The default value is true, indicating that the IBM App Connect Enterprise
 WSRR cache is enabled.
```

10. Edit the *predefinedCacheQueries* parameter in the `.conf.yaml` file to show a list of semicolon-separated WSRR XPath query expressions with which to initialize the IBM App Connect Enterprise cache at start-up.

```
predefinedCacheQueries: "//*[@name='Wss100.wsdl'];/WSRR/WSDLService/ports[binding/
portType[@name='DemoCustomer' and @namespace='http://demo.sr.eis.ibm.com'] and
 @PortProperty1='ValueOfPortProperty1' and exactlyClassifiedByAllOf(., 'http://www.ibm.com/
xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState1','http://
www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#Approve')]" #  A list
 of semicolon-separated WSRR XPath query expressions with which to initialize the IBM App
 Connect Enterprise WSRR cache at start-up.
```

    These WSRR XPath query expressions are defined by the WSRR SOAP interface query expression language, and might include an optional depth specifier extension. For more information about XPath, see https://www.ibm.com/support/knowledgecenter/en/SSWLGF_8.5.6/com.ibm.sr.doc/twsr_xpath_query.html.

11. Set the *refreshQueriesAfterNotification* parameter in the `.conf.yaml` file to `true` or `false` according to your requirements.

    The default value is `true`. When a notification is received from WSRR, and *refreshQueriesAfterNotification* is set to `true`, the Cache is updated with the new version of the object immediately. If it is set to `false`, the Cache is updated on the next request.

```
refreshQueriesAfter Notification: true # When a notification is received from WSRR, if
 refreshQueriesAfterNotification is set to true, the Cache is updated with the new version
 of the object immediately; if it is set to
                                                    false, the Cache is updated on the next
 request. The default value is true.
```

12. Edit the *subscriptionTopic* parameter in the `.conf.yaml` file to show topic name that is used to receive the WebSphere Application Server JMS provider Cache Notifications. The default value is `'jms/SuccessTopic'`.

```
subscriptionTopic: 'jms/SuccessTopic' # The topic name that is used to receive WebSphere
 Application Server JMS provider Cache Notifications. The default value is 'jms/
SuccessTopic'
```

13. Edit the *timeout* parameter in the `.conf.yaml` file to show the cache expiry time in milliseconds. The default value is `100000000`, which is approximately 27.8 hours.

```
timeout: 100000000 # The cache expiry time in milliseconds. The default value is 100000000,
 which is approximately 27.8 hours.
```

14. Restart the integration node or independent integration server. For more information, see “Starting and stopping an integration node” on page 247, “Starting an integration server” on page 250, and “Stopping an integration server” on page 254.

Accessing a secure WebSphere Service Registry and Repository repository

To access a secure WebSphere Service Registry and Repository repository, edit the parameters in the `.conf.yaml` configuration file for the integration node, or integration server.

About this task

You must connect over HTTPS, not HTTP, which is specified in the `endpointAddress` property of the `WSSRConnectorProvider` entry in the `ConnectorProviders` section of the `node.conf.yaml` or `server.conf.yaml` configuration file. The configuration must specify the domain qualified hostname and port of your WebSphere Service Registry and Repository server and you must define the WebSphere Service Registry and Repository server logon credentials and HTTPS SSL certificates.

For more information about the `endpointAddress` configuration parameter, see "Configuration parameters for the WebSphere Service Registry and Repository nodes" on page 1006.

To access a secure WebSphere Service Registry and Repository from a managed server, enter the following sequence of commands. Any properties that you set for an integration node, in the `node.conf.yaml` file, are inherited by the integration servers that it owns. However, you can change the owned integration server properties by modifying them in the appropriate `server.conf.yaml` file. For an independent integration server, edit the `server.conf.yaml` configuration file to set the properties that would be set by the following commands. For more information about configuring integration servers, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Procedure

1. Ensure that the integration node is running. If it is not, use the **mqsistart** command to start it.
2. Edit the `.conf.yaml` file to configure the integration node or integration server to use HTTPS to communicate with the WebSphere Service Registry and Repository server.

   ```
   WSRRConnectorProvider:

             # endpointAddress                              # The location or endpoint of the
    WSRR server.
   ```

   The default value for all versions of WebSphere Service Registry and Repository is `https://fill.in.your.host.here:9080/WSRRCoreSDO/services/WSRRCoreSDOPort`

   For more information about the specific levels of WebSphere Service Registry and Repository that are supported by IBM App Connect Enterprise, see IBM App Connect Enterprise system requirements web page.

3. Configure the integration node keystore to contain your WebSphere Service Registry and Repository server certificate keys; for a discussion of digital certificates, see "Digital certificates" on page 2493. Obtain these certificate keys from the installation of the WebSphere Application Server that hosts your WebSphere Service Registry and Repository server. The integration node uses a single keystore. Therefore, if your integration node also implements WS-Security, HTTPS, or SSL-secured IBM MQ, you might need to merge the provided keys into an existing keystore file. Display the current configuration parameters of the integration node by using the following command:

   ```
   mqsireportproperties INODE -o BrokerRegistry -r
   ```

   The parameters in this example represent the following values:

   The **-o** parameter specifies the name of the object (in this case, `BrokerRegistry`)

The **-r** parameter specifies that all property values of the object are displayed, including the child values, if appropriate.

If your WebSphere Service Registry and Repository server uses mutual SSL authentication, configure the integration node keystore to contain a private key. Then, set the `brokerKeystoreFile` configuration parameters for the integration node, by using one of the following options.

- Use the **mqsichangeproperties** command to change configuration parameters for the integration node.

```
mqsichangeproperties INODE -o BrokerRegistry
-n brokerKeystoreFile -v C:\WSRR\SSL\ClientKeyFile.jks
```

  The parameters in this example represent the following values:

  The **-o** parameter specifies the name of the object (in this case, `BrokerRegistry`).
  The **-n** parameter specifies the names of the properties to be changed (in this case, `brokerKeystoreFile`).
  The **-v** parameter specifies the values of properties that are defined by the **-n** parameter (in this case, `C:\WSRR\SSL\ClientKeyFile.jks`).

- Edit the `.conf.yaml` file to change configuration parameters for the integration node.

```
BrokerRegistry:
           brokerKeystoreType: 'JKS'                        # Trust store type
           brokerKeystoreFile: 'C:\devel\ACE11\WSRR
\comics1.fyre.ibm.com.serverKeyFile.jks'      # Location of the broker key store
         # Location of the broker trust key store
```

4. Configure the integration node truststore to contain signer certificates for your WebSphere Service Registry and Repository server. As stated for the keystore, the integration node uses a single truststore. Therefore, you might need to merge certificates into an existing truststore file. The integration node truststore is configured by using the **mqsichangeproperties** command. To change the `brokerTruststoreFile` configuration parameters for the integration node, use the following command:

```
mqsichangeproperties INODE -o BrokerRegistry
-n brokerTruststoreFile -v C:\WSRR\SSL\ClientTrustFile.jks
```

   The parameters in this example represent the following values:

   The **-o** parameter specifies the name of the object (in this case, `BrokerRegistry`).
   The **-n** parameter specifies the names of the properties to be changed (in this case, `brokerTruststoreFile`).
   The **-v** parameter specifies the values of properties that are defined by the **-n** parameter (in this case, `C:\WSRR\SSL\ClientTrustFile.jks`).

5. Stop the integration node by using the **mqsistop** command.

   You must stop the integration node to complete the following step.

6. If you are using the secure credentials vault to store the WebSphere Service Registry and Repository server credentials, complete the following steps:

   Create a vault key for the specified server by running the **mqsivault** command:

```
mqsivault INODE --create --vault-key 12345678
```

   The parameters in this example represent the following values:

   The `--create` parameter creates a vault for the specified server.

The `--vault-key` parameter specifies the vault key to be used for creating the vault. (in this case, 12345678).

Define the WebSphere Service Registry and Repository type default credential-name in the `node.conf.yaml` configuration file by running the following command:

```
mqsicredentials INODE --all-integration-servers --set-as-default --credential-type wsrr  --
credential-name wsrrServerCred
```

For an independent integration server, add the following lines to the `server.conf.yaml` configuration file.

```
Defaults:
  Credentials:
    wsrr: 'wsrrServerCred'
```

7. Set the WebSphere Service Registry and Repository server username and password by using one of the following methods:
   - Set the username and password by running the **mqsisetdbparms** command.

     ```
     mqsisetdbparms INODE -n DefaultWSRR::WSRR -u wasuser -p waspass
     ```

     The parameters in this example represent the following values:

     The **-n** parameter specifies the name of the data source (in this case, `DefaultWSRR::WSRR`).
     The **-u** parameter specifies the user ID to be associated with this data source (in this case, `wasuser`).
     The **-p** parameter specifies the password to be associated with this data source (in this case, `waspass`).
   - Create the username and password by running the **mqsicredentials** command:

     ```
     mqsicredentials INODE --all-integration-servers --vault-key <myvaultkey> --create --
     credential-type wsrr --credential-name wsrrServerCred wsrr --username <wsrr user> --
     password <wsrr password>
     ```

     The parameters in this example represent the following values:

     The **--all-integration-servers** parameter creates a vault for all servers. (use –integration-server for a specified server).
     The **--vault-key** parameter specifies the vault key to be used for creating the vault.
     The **--create** parameter specifies the vault key to be used for creating the vault.
     The **--credential-type** parameter specifies the type of credential (in this example, `wsrr` is used).
     The **--username** parameter specifies the username.
     The **--password** parameter specifies the password.
8. If your WebSphere Service Registry and Repository server uses mutual SSL authentication, set the `brokerKeystore` username and password by using one of the following methods.
   - Set the username and password by running the **mqsisetdbparms** command:

     ```
     mqsisetdbparms INODE -n brokerKeystore::password -u dummy -p WebAS
     ```

     The parameters in this example represent the following values:

     The **-n** parameter specifies the name of the data source (in this case, `brokerKeystore::password`).
     The **-u** parameter specifies the user ID to be associated with this data source (in this case, `dummy`).

The **-p** parameter specifies the password to be associated with this data source (in this case, WebAS).

- Set the brokerKeyStore by running the **mqsicredentials** command:

```
mqsicredentials INODE --all-integration-servers --vault-key <myvaultkey> --create --
credential-type keystore --credential-name password --password changeme
```

The parameters in this example represent the following values:

The **--all-integration-servers** parameter specifies that the command applies to all integration servers on the specified integration node. Alternatively, you can specify a named integration server (`-integration-server IntegrationServerName`).
The **--vault-key** parameter specifies the vault key that was used to create the vault.
The **--create** parameter creates credentials in the vault.
The **--credential-type** parameter specifies the credential type.
The **--credential-name** parameter specifies the name of the credential. In this case, `password` is used. You must update the `node.conf.yaml` to set this name as follows.

```
BrokerRegistry:
brokerTruststorePass: ‘password'
```

The **--password** parameter specifies the password to be associated with this resource.

9. Set the `brokerTrustStore` username and password by using one of the following methods:

- Set the username and password by running the **mqsisetdbparms** command.

```
mqsisetdbparms INODE -n brokerTruststore::password -u dummy
-p WebAS
```

The parameters in this example represent the following values:

The **-n** parameter specifies the name of the data source (in this case, `brokerTruststore::password`).
The **-u** parameter specifies the user ID to be associated with this data source (in this case, `dummy`).
The **-p** parameter specifies the password to be associated with this data source (in this case, `WebAS`).

- Set the `brokerTrustStore` by running the **mqsicredentials** command:

```
mqsicredentials INODE --all-integration-servers --vault-key <myvaultkey> --create --
credential-type truststore --credential-name password --password changeme
```

You must update the `node.conf.yaml` file set the specified credential name as follows.

```
BrokerRegistry:
  brokerTruststorePass: 'password'
```

The parameters in the example represent the following values:

The **--all-integration-servers** parameter specifies that the command applies to all integration servers on the specified integration node. Alternatively, you can specify a named integration server (`-integration-server IntegrationServerName`).
The **--vault-key** parameter specifies the vault key that was used to create the vault.
The **--create** parameter creates credentials in the vault.
The **--credential-type** parameter specifies the credential type.
The **--credential-name** parameter specifies the name of the credential. In this case, `password` is used to match the setting of **BrokerRegistry.brokerKeystorePass** in the `.conf.yaml` configuration file.
The **--password** parameter specifies the password to be associated with this resource.

10. To use cache notification with your secure WebSphere Service Registry and Repository server, follow the instructions in “Setting up cache notification” on page 1016.

11. Restart the integration node by using the **mqsistart** command.

    If you create a vault key, you must specify the **--vault-key** parameter in the **mqsistart** command.

    ```
    mqsistart INODE --vault-key 12345678 -u userid -p password
    ```

Caching artifacts from the WebSphere Service Registry and Repository

IBM App Connect Enterprise saves the data that it retrieves from the WebSphere Service Registry and Repository (WSRR) in a local cache.

The WSRR nodes, EndpointLookup and RegistryLookup, can retrieve data that was stored in the Integration node WSRR cache by a previous query, improving performance and message throughput. The first occurrence of each query is always sent to WSRR. By default, this activity occurs when a WSRR node first issues a specific query, although it is possible to pre-populate the cache, when the integration node or integration server starts, by using the queries described here.

Configuring the WSRR Cache

The cache is configured individually for each integration node or independent integration server by editing the parameters in the `.conf.yaml` configuration file; for more information, see “Configuration parameters for the WebSphere Service Registry and Repository nodes” on page 1006.

You can configure the cache in the following ways:

- **Disabling the cache**

  Disable the cache by setting the `needCache` parameter to `false`. By default, the cache is enabled, but the WSRR nodes can operate without the cache. If the cache is disabled, every query that is issued by the node is sent to WSRR, ensuring that the results of the query always reflect the current contents of the registry. This activity can affect performance.

- **Preloading the cache**

  Preload the cache by setting the `predefinedCacheQueries` parameter. By default, no items are preloaded in the cache and the first occurrence of every query is sent to WSRR. You can specify predefined queries that are executed when the integration node or integration server starts, or when a message flow that contains WSRR flows is first deployed, populating the cache for use by subsequent WSRR nodes. By specifying predefined queries, performance might be affected at startup, rather than on the first occurrence of a query at run time. The predefinedCacheQueries parameter is a list of WSRR XPath query expressions that are separated by semicolons, each with an optional depth specification. The user trace shows the WSRR XPath query expression that is generated by a WSRR node.

- **Changing the cache expiry timeout value**

  Change the cache expiry timeout value by setting the `timeout` parameter. The cached results of a query are discarded after the specified elapses. The next occurrence of the query is sent to WSRR and the new result is entered in the cache. If the contents of the registry are likely to change frequently, you can specify a shorter expiry timeout value so that changes are picked up quicker. This activity affects performance as more queries are sent to WSRR.

- **Enabling cache notification**

  Enable cache notification by setting the `enableCacheNotification` parameter to `true` and by setting the `initialContextFactory` and `locationJNDIBinding` properties for your WSRR server. By default, cache notification is disabled. Cache notification is a more flexible method than expiry timeout for refreshing cached data because it allows individual WSRR entities to be refreshed at the time they are modified in WSRR.

  If cache notification is enabled, the cache subscribes to events that occur in WSRR and is notified when an object is updated or deleted in WSRR. The object is discarded from the cache. If the `refreshQueriesAfterNotification` parameter is set to `true`, the cache is updated with the new version of the object immediately. If the `refreshQueriesAfterNotification` parameter is set to `false`, the cache is updated the next time that a relevant query is issued by a WSRR node.

Setting up cache notification
Edit the `.conf.yaml` configuration file to enable cache notification so that the cache is notified of events that occur in WebSphere Service Registry and Repository (WSRR).

About this task

WSRR publishes notification events by using WebSphere Application Server. Cache notification allows the cache to subscribe to these events.

To set the WSRR parameters, you must edit the `WSRRConnectorProvider` entries in the `ConnectorProviders` section of the `node.conf.yaml` or `server.conf.yaml` configuration file. When you create a new integration node or a new independent integration server, the `.conf.yaml` configuration file contains the following subset of the WSRR parameters. All the other WSRR parameters have default settings which generally do not need to be changed. If you need to change any of the other WSRR parameters, you must add them to the `.conf.yaml` configuration file.

```
ConnectorProviders:
    #WSRRConnectorProvider:    # Requires the optional WSRR component install
    #endpointAddress: 'https://host:9443/WSRR8_0/services/WSRRCoreSDOPort'   # WSRR server
 endpoint url
    #needCache: true                                                        # enable WSRR cache
    #predefinedCacheQueries: ''                                             # semicolon-
separated XPath querys to initialize WSRR cache at start-up
    #enableCacheNotification: false                                         # enable WSRR cache
 notification
    #locationJNDIBinding:  'iiop://host:2809'                               # WSRR cache WAS
 JMS provider JNDI bindings url
```

To enable cache notification, complete the following steps to change the relevant properties in the `.conf.yaml` configuration file, and to add a user ID and password if you are connecting to a secure WebSphere Application Server. Any properties that you set for the integration node, in the `node.conf.yaml` file, are inherited by the integration servers that it owns. However, you can change any of the integration server properties by modifying them in the appropriate `server.conf.yaml` file. For more information about configuring integration servers that are managed by an integration node, see Configuring an integration server by modifying the server.conf.yaml file.

Procedure

1. Set the **enableCacheNotification** property value to `true` in the `.conf.yaml` configuration file:

   ```
   enableCacheNotification: true # The default value is false. Select true to enable IBM® App
    Connect Enterprise WSRR Cache Notification.
   ```

2. Set the **locationJNDIBinding** property to the value that you require for your WSRR server in the `.conf.yaml` configuration file:
   For example:

   ```
   locationJNDIBinding: 'iiop://host_name:2809/' # The URL to the WebSphere Application Server
    JMS provider JNDI bindings. The default value is iiop://host_name:2809/
   ```

3. When you use WebSphere Application Server Version 8.0 or later, the default IIOP secure setting is `SSL-Required`. Therefore, if you connect to your server without changing this option to `SSL-Optional`, you must configure the `sas.client.props` and the `ssl.client.props` files in the following way:

   `sas.client.props`

   ```
   - Set com.ibm.ssl.alias=AnotherSSLSettings
   - Set com.ibm.CORBA.loginSource=properties
   ```

   `ssl.client.proprs`

   ```
   -- Use the AnotherSSLAlias template that is already in the file, but comment ou the
    DefaultAlias Keystore and Truststore information to prevent it affecting the current
    configuration
   ```

```
-- Comment out the KeyStore part of the 'another SSL Alias' because the TrustStore is the
 only one that is required:
- Set com.ibm.ssl.enableSignerExchangePrompt=false in all scenarios
```

4. Set the **secureJNDIPropertiesFiles** property in the `.conf.yaml` file to point to the directory where your `.props` files are stored.
   For example:

```
secureJNDIPropertiesFiles: 'C:\SSLPropsFiles' #  The directory where your .props files are
 stored. For example C:\SSLPropsFiles.
```

If you are connecting to a secure WebSphere Application Server, you must use a user ID and password. To set the user ID and password complete one of the following steps:

5. Optional: Set the user ID and password by using the **mqsisetdbparms** command:
   a) If the integration node is running, stop it by using the **mqsistop** command.
   b) Issue the **mqsisetdbparms** command to set up your user ID and password.
      For example:

```
mqsisetdbparms INODE -n jms::DefaultWSRR@jms/SRConnectionFactory
-u userid -p password
```

      where:

      -n specifies the name of the data source
      (in this case, `jms::DefaultWSRR@jms/SRConnectionFactory` is used.
      You can also use `wsrr`)
      -u specifies the user ID to be associated with this data source
      (in this case, *userid*)
      -p specifies the password to be associated with this data source
      (in this case, *password*)
   c) Restart the integration node by using the **mqsistart** command.
6. Optional: Set the user ID and password by using the **mqsicredentials** command:
   a) If the integration is running, stop it by using the **mqsistop** command.
   b) Issue the **mqsicredentials** command to set up your user ID and password.
      For example:

```
mqsicredentials --work-dir --create --credential-type jms --credential-name wsrr@jms/
SRConnectionFactory --vault-key 12345abcde --username jmsvaultuser --password jmsvaultpwd
-u userid -p password
```

      where:

      --work-dir specifies path to the work directory that is used by an independent integration server.

      --credential-type specifies the resource that requires credentials for access.
      (in this case, *jms*)
      --credential-name specifies the vault key that is used to access the vault where the credential is stored.
      --vault-key specifies the vault key that is used to access the vault where the credential is stored.
      --username specifies the user ID to be associated with this resource.
      --password specifies the password to be associated with this resource.
   c) Restart the integration node by using the **mqsistart** command.

Dynamically defining the search criteria

You can use the RegistryLookup and EndpointLookup nodes to issue WebSphere Service Registry and Repository (WSRR) queries specified in the local environment.

The RegistryLookup and EndpointLookup nodes issue WSRR queries at run time and save the resulting data in the local environment. You can specify the queries at design time by using node properties to

define the search criteria. Both nodes require at least one query property to be defined before you can deploy the message flow. However, you can specify the search criteria at run time in the local environment, either supplementing or overriding the node properties.

The following table defines the local environment overrides for WSRR queries. These fields must be set in OutputLocalEnvironment.ServiceRegistryLookupProperties by a preceding transformation node, such as a Compute node.

Setting	Description
Name	This setting overrides the Name property on the node; for example, with an ESQL Compute node: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Name = 'DemoCustomer';` This setting relates to the PortType Name property on the EndpointLookup node. Therefore, to set the PortType Name property, use the Name setting in the local environment.
Namespace	This setting overrides the Namespace property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Namespace = 'http://mb.sr.eis.ibm.com';` This setting relates to the PortType Namespace property on the EndpointLookup node. Therefore, to set the PortType Namespace property, use the Namespace setting in the local environment.
Version	This setting overrides the Version property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Version = '1.0';` This setting relates to the PortType Version property on the EndpointLookup node. Therefore, to set the PortType Version property, use the Version setting in the local environment.
MatchPolicy	This setting overrides the Match Policy property on the node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.MatchPolicy = 'One';` Valid values are One and All.

Setting	Description
DepthPolicy	This setting overrides the `Depth Policy` property on the RegistryLookup node; for example: `SET OutputLocalEnvironment.ServiceRegistryLookupProperties.DepthPolicy = 'MatchOnly';` Valid values are: • `MatchOnly` for `Return matched only (Depth = 0)` • `MatchShowRel` for `Return matched showing immediate relationships (For compatibility only)` • `MatchPlusImmediate` for `Return matched plus immediate related entities (Depth = 1)` • `MatchPlusAll` for `Return matched plus all related entities (Depth = -1)` The `MatchShowRel` property provides compatibility with versions of IBM App Connect Enterprise before Version 6.1.0.4, by using the output format that was used in those previous versions. This option is deprecated, and should not be used if you are creating a new message flow. Consider migrating existing message flows to use one of the other options.

UserProperties	This setting overrides the `User Properties` property on the node. You can specify more than one user-defined property in the local environment; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = 'value1';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property2 = 'value2';`
	You can remove a user-defined property from the local environment by setting its value to NULL; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.UserProperties.property1 = NULL;`
	You can use the node properties editor at design time to specify ESQL paths or XPath expressions to read the value for a user property at run time from a field in the message tree. However, the override values that you set in the local environment are the string values that are used in the query.

Classification	This setting overrides the `Classification` property on the node; for example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	You can specify more than one classification in the local environment. For example:
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[1] = 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/DefaultLifecycle#InitialState0';`
	`SET OutputLocalEnvironment.ServiceRegistryLookupProperties.Classification[2] = 'http://www.ibm.com.policy/GovernancePolicyDomain';`

EndpointLookup node output

Use an EndpointLookup node to retrieve the endpoint addresses for WSDL service definitions held in WebSphere Service Registry and Repository (WSRR).

The input message is not changed by the EndpointLookup node. Instead, the local environment is updated to contain details of the endpoints retrieved by the query specified by the node and any local environment overrides.

You can configure the EndpointLookup node to dynamically set the service endpoint address for services that will be invoked by a subsequent SOAP or HTTP request node. The EndpointLookup node sets the destination URL in the local environment overrides for those nodes.

EndpointLookup node output if the `Match Policy` property is set to `One`

If the `Match Policy` property of the node is set to `One`, theEndpointLookup node inserts the endpoint URL retrieved by the query into the local environment in an ITService entry in the local environment under ServiceRegistry, and sets the destination overrides for the SOAP and HTTP request nodes that can be connected directly to its output terminal. The following locations are updated:

- `LocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL`
- `LocalEnvironment.Destination.HTTP.RequestURL`

These settings override the `Web service URL` property of the SOAPRequest, SOAPAsyncRequest, and HTTPRequest nodes, allowing a dynamic call to a web service provider.

The following example shows typical output from the EndpointLookup node when the `Match Policy` is set to `One`. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <Destination>
    <SOAP>
      <Request>
        <Transport>
          <HTTP>
            <WebServiceURL>http://localhost:9081/DemoCustomerWeb/
              services/DemoCustomer</WebServiceURL>
          </HTTP>
        </Transport>
      </Request>
    </SOAP>
    <HTTP>
      <RequestURL>http://localhost:9081/DemoCustomerWeb/
        services/DemoCustomer
    </RequestURL>
   </HTTP>
  </Destination>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

EndpointLookup node output if the Match Policy property is set to All

If the Match Policy is set to All the EndpointLookup node writes an ITService entry in the local environment location ServiceRegistry for each endpoint retrieved by the query.

The following example shows typical output from the EndpointLookup node when the Match Policy is set to All. (Other entries might exist in the local environment depending on previous processing in the flow.)

```
<LocalEnvironment>
  <ServiceRegistry>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer</Address>
        <PortType>
          <name>DemoCustomer</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
        <Property>
          <name>policy</name>
          <value>RM</value>
        </Property>
        <Property>
          <name>country</name>
          <value>China</value>
        </Property>
        <Classification>http://eis.ibm.com/ServiceRegistry/
          GenericObjecttypes#Routing</Classification>
      </Endpoint>
    </ITService>
    <ITService>
      <Endpoint>
        <Address>http://localhost:9081/DemoCustomerWeb/
          services/DemoCustomer2</Address>
        <PortType>
          <name>DemoCustomer2</name>
          <namespace>http://demo.sr.eis.ibm.com</namespace>
          <version>1.0</version>
        </PortType>
      </Endpoint>
    </ITService>
  </ServiceRegistry>
</LocalEnvironment>
```

RegistryLookup node output

Use the RegistryLookup node to retrieve any type of entity held in the WebSphere Service Registry and Repository (WSRR). The entities that match the specified search criteria are stored in the local environment. The input message is not modified.

This topic contains the following sections:

Setting the Depth Policy property

The Depth Policy property on the RegistryLookup node specifies how much data is returned for each matched entity. The following table shows the valid values.

Depth Policy property value	Local environment override value	Data returned
Return matched only (Depth = 0)	MatchOnly	Just the matched entities

Depth Policy property value	Local environment override value	Data returned
Return matched showing immediate relationships (For compatibility only)	MatchShowRel	The matched entities and additional references
Return matched plus immediate related entities (Depth = 1)	MatchPlusImmediate	The matched entities and the immediate related child entities
Return matched plus all related entities (Depth = -1)	MatchPlusAll	The matched entities and all the related child entities

Use MatchShowRel for compatibility with versions of WebSphere Message Broker before Version 6.1.0.4. The local environment output for MatchShowRel is shown in Example 1 and follows the format that was used in those previous versions. However, the MatchShowRel option is deprecated, and provided only for compatibility with previous versions. Do not use MatchShowRel if you are creating a new message flow, and consider migrating existing message flows to use one of the other options.

Use the MatchOnly option to retrieve just the individual entities matched by the search criteria. This option is efficient, however the local environment output does not contain any information about entities related to the matched entities.

Use the MatchPlusImmediate option to retrieve the entities matched by the search criteria, and the related child entities. This option offers a useful compromise, allowing you to access the immediate relations of the matched entities, while still restricting the total amount of data retrieved.

Use one of the MatchOnly or MatchPlusImmediate options when migrating a message flow that uses the deprecated MatchShowRel option. If your original message flow used the relationship information in the matched entities, use the MatchPlusImmediate option. See "Migrating a message flow that uses the MatchShowRel option" on page 1024.

Use the MatchPlusAll option to retrieve the entities matched by the search criteria, and all the related child entities. Use this option only if your message flow needs access to more than the immediate relations of the matched entities, because it retrieves considerably more data than the MatchPlusImmediate option, see "Local environment output tree" on page 1022.

Local environment output tree

The local environment output tree has a different format when the deprecated MatchShowRel option of the Depth Policy property is used. The following table describes the differences in the local environment output tree format.

MatchOnly, MatchPlusImmediate, and MatchPlusAll options	MatchShowRel option
The ServiceRegistry folder element is owned by the XMLNSC compiler.	No owning parser on the ServiceRegistry folder element, and each Entity element is owned by the XMLNS parser
The ServiceRegistry tree does not use unnecessary namespaces.	Namespaces are attached to all UserDefined folder elements, meaning that the path specified must declare and use the relevant namespace to access fields within these folders.
The ServiceRegistry tree is optimized through use of the XMLNSC parser.	The output tree contains a number of XML declaration, pcdata, and white space elements that have no business significance.

MatchOnly, MatchPlusImmediate, and MatchPlusAll options	**MatchShowRel option**
In WSRR, binary data is represented as a GenericDocument with a `content` attribute. The `content` attribute is represented in the local environment as XMLNSC type Attribute +base64Binary, (0x03000160). You can access the unencoded binary data directly as `Entity.content`. However, because of the special XMLNSC element type, the data is automatically base64 encoded if the tree is serialized.	Binary content is represented as a base64 encoded character string in the `content` attribute. The message flow must invoke a Java method to decode the value if the original binary data is required.
The retrieved entities are not modified to add any user properties.	The matched Entities appear in the local environment with a user property called `WSRREncoding`. This property has no specific significance to message flow processing. If there is a user property called `WSRREncoding` defined for the entity, the defined value is used, otherwise the `WSRREncoding` property is added with `value="DEFAULT"`.

The `MatchPlusImmediate` and `MatchPlusAll` options result in a graph of entities being returned when WSRR executes the query. Each entity in the graph can refer to other entities. There are two types of relationship that cannot be expressed directly in a hierarchical tree:

- Cyclic references - a graph can contain an entity from which you can follow relationships that arrive back at the same entity.
- Multiple references - a graph can contain an entity that is referenced by multiple entities.

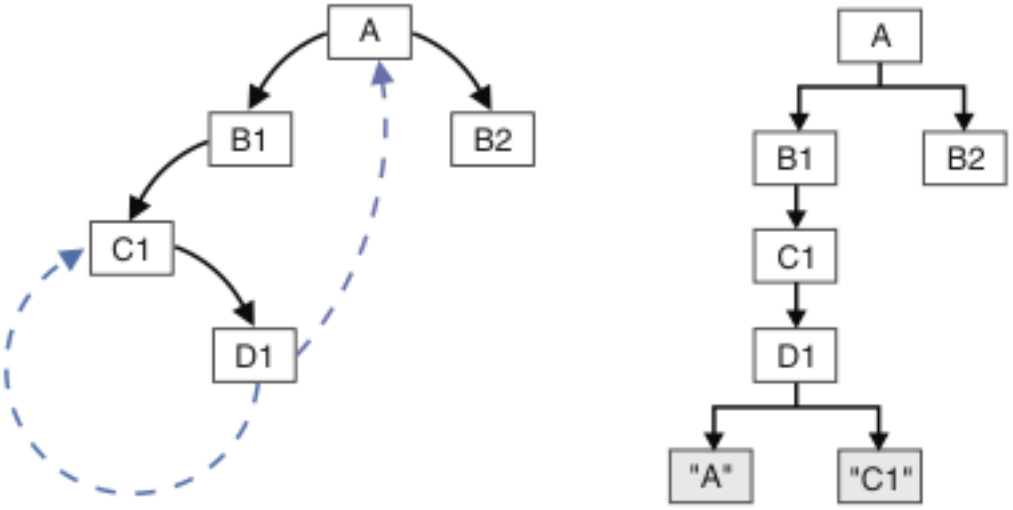
The representation of these relationships in the local environment output tree is described in the following text.

The WSRR graph is represented in the local environment output tree as follows:

- The matched entities that are returned in the WSRR graph are represented as Entity elements as the first children of the LocalEnvironment.ServiceRegistry. The properties of the matched entities are represented as child elements.
- A matched entity can contain references to other entities. If `MatchPlusImmediate` is set, these references are represented as Entity child elements of the matched entity. If `MatchPlusAll` is set, the same rule is applied recursively to these children.
- Cyclic references - if an entity references another entity that is one of its ancestors in the WSRR graph, the referenced entity is represented as an EntityRef element. An EntityRef element does not represent

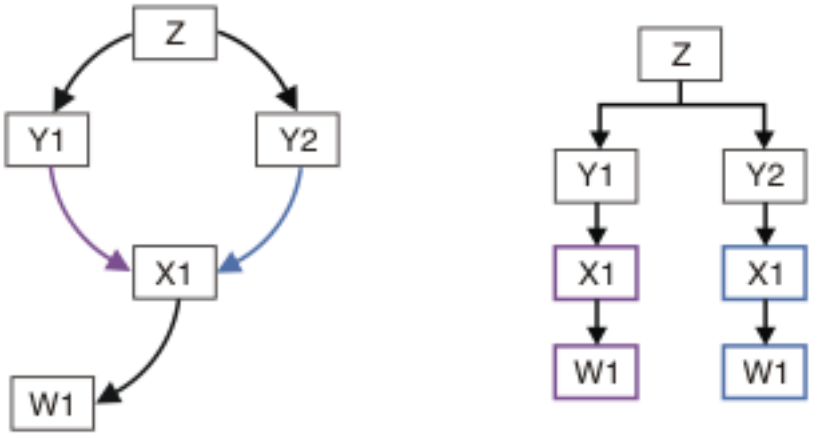
the target entity directly, but provides information to identify it. This prevents the possibility of entering cyclic loops when navigating the tree.

The following diagram shows references from entity D1 in the graph to entities C1 and A that are ancestral; therefore, Entity D1 in the message tree contains EntityRef elements for A and C1.



- Multiple references - if an entity in the WSRR SDO graph is referenced by more than one other entity, it is represented as a separate Entity element in the message tree each time it is referenced. The Entity element is cloned, along with any entities that it refers to.

  The following diagram shows that entity X1 in the graph is referenced by entities Y1 and Y2, so Entity X1 and Entity W1 that it references are modeled twice in the message tree.



Migrating a message flow that uses the `MatchShowRel` option

When you migrate an existing message flow that uses the deprecated `MatchShowRel` option of the `Depth Policy` property, the local environment output tree will have a different format. The table shown earlier describes the differences in the local environment output tree formats, and indicates what you must modify in the message flow to access the data in the updated format; see “Local environment output tree” on page 1022.

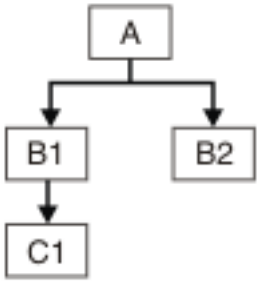
Examples

The following examples show typical output from the RegistryLookup node:

- Example 1 shows the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)`, and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output. See "RegistryLookup node output: example 1" on page 1025
- Example 2 shows the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities. See "RegistryLookup node output: example 2" on page 1028
- Example 3 shows the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)`. See "RegistryLookup node output: example 3" on page 1031.

RegistryLookup node output: example 1
Example showing the full RegistryLookup node output in two cases for a query that returns two versions of a concept entity. In both cases the `Match Policy` property is set to `All`. In the first case the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)`, and in the second case the `Depth Policy` property is set to `Return matched only, showing immediate relationships (For compatibility only)`. This example also shows example ESQL to read elements of the output.

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when an entity called ConceptA1 is retrieved from WebSphere Service Registry and Repository (WSRR). ConceptA1 is defined in WSRR with 2 versions: 1.0 which is deprecated, and 2.0 which is classified as initial state. Additional properties and relationships have been added to the 2.0 version. The following diagram shows the relationships between the elements in the message tree.



The following message trees show the different values of the `Depth Policy` property.

- `Depth Policy` property set to `Return matched plus immediate related entities (Depth = 1)`

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

```
SET OutputRoot.XMLNSC.Result.ServiceRegistry =
  InputLocalEnvironment.ServiceRegistry;
```

  The ServiceRegistry folder element is owned by the XMLNSC parser so you can invoke a like parser tree copy to OutputRoot.XMLNSC. The following XML, which has been formatted, is produced when writing this output root tree.

```
<ServiceRegistry>
  <Entity
    bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="2.0"
    description="A version 2 of the existing one"
    owner="UNAUTHENTICATED"
    lastModified="1230116323343"
    creationTimestamp="1227176757406"
```

```
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#InitialState0
    </classificationURIs>
    <classificationURIs>
      http://www.ibm.com.policy/GovernancePolicyDomain
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1 for v1" />
    <userDefinedProperties name="property2" value="value1 for v2" />
    <userDefinedProperties name="property3" value="value1 for v3" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="8203cb82-8827-4757.99e1.36de6036e1af"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1"
          version="2.0"
          description="Next revision of this concept"
          owner="UNAUTHENTICATED"
          lastModified="1227191748250"
          creationTimestamp="1227177460156"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing"
          owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
    <userDefinedRelationships name="ReferTo">
      <targetEntities>
        <Entity
          bsrURI="81e45381-a9be-4ea4.9519.53657953196d"
          name="ConceptC1"
          namespace="http://www.examples.com/ConceptC1"
          version="1.0"
          description="Test stuff C1"
          owner="UNAUTHENTICATED"
          lastModified="1227874855140"
          creationTimestamp="1227177519609"
          lastModifiedBy="UNAUTHENTICATED" primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
  <Entity
    bsrURI="b68952b6-8d68-4840.8e5e.a3716da35e2e"
    name="ConceptA1"
    namespace="http://www.examples.com/ConceptA1"
    version="1.0"
    description="The original concept"
    owner="UNAUTHENTICATED"
    lastModified="1229030287593"
    creationTimestamp="1227173773250"
    lastModifiedBy="UNAUTHENTICATED"
    primaryType="">
    <classificationURIs>
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </classificationURIs>
    <userDefinedProperties name="property1" value="value1" />
    <userDefinedProperties name="property2" value="value2" />
    <userDefinedRelationships name="ContainsChildren">
      <targetEntities>
        <Entity
          bsrURI="7d5fd37d-e90a-4ab0.89eb.d25b81d2ebec"
          name="ConceptB1"
          namespace="http://www.examples.com/ConceptB1" version="1.0"
          description="" owner="UNAUTHENTICATED"
          lastModified="1227874785062"
          creationTimestamp="1227177401265"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
        <Entity
          bsrURI="a0d2bba0-f395-45bc.929e.04d143049eb5"
```

```
          name="ConceptB2"
          namespace="http://www.examples.com/ConceptB2" version="1.0"
          description="Testing" owner="UNAUTHENTICATED"
          lastModified="1227191700812"
          creationTimestamp="1227177334515"
          lastModifiedBy="UNAUTHENTICATED"
          primaryType="" />
      </targetEntities>
    </userDefinedRelationships>
  </Entity>
</ServiceRegistry>
```

The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the `Depth Policy` property is set to `Return matched plus immediate related entities (Depth = 1)`.

```
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].
  userDefinedProperties[1].value;
```

- `Depth Policy` property set to `Return matched showing immediate relationships (For compatibility only)`

  The following ESQL was used to create a serialization of the output of the ServiceRegistry node:

  ```
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputLocalEnvironment.ServiceRegistry.Entity[]);
  WHILE I < J DO
    SET OutputRoot.XMLNS.ServiceRegistry.Entity[I] =
      InputLocalEnvironment.ServiceRegistry.Entity[I];
    SET I = I + 1;
  END WHILE;
  ```

  The ServiceRegistry folder does not have an owning parser so you must navigate to the ""elements and initiate a like-parser-copy to the XMLNS owned output root tree. The following XML is produced when writing this output root tree.

  ```
  <ServiceRegistry>
    <Entity
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sdo="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo"
      xsi:type="sdo:GenericObject"
      bsrURI="33a9ad33-d4a4-442e.b3a6.37e62137a605"
      name="ConceptA1"
      namespace="http://www.examples.com/ConceptA1"
      version="2.0"
      description="A version 2 of the existing one"
      owner="UNAUTHENTICATED"
      lastModified="1229439847694"
      creationTimestamp="1227176757406"
      lastModifiedBy="UNAUTHENTICATED"
      primaryType="">
      <sdo:classificationURIs>
        http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
          DefaultLifecycle#InitialState0
      </sdo:classificationURIs>
      <sdo:classificationURIs>
        http://www.ibm.com.policy/GovernancePolicyDomain
      </sdo:classificationURIs>
      <sdo:userDefinedRelationships
        name="ContainsChildren"
        targets="8203cb82-8827-4757.99e1.36de6036e1af
          a0d2bba0-f395-45bc.929e.04d143049eb5" />
      <sdo:userDefinedRelationships
  ```

```
      name="ReferTo"
      targets="81e45381-a9be-4ea4.9519.53657953196d" />
    <sdo:userDefinedProperties name="property1" value="value1 for v2" />
    <sdo:userDefinedProperties name="property2" value="value2 for v2" />
    <sdo:userDefinedProperties name="property3" value="value3 for v2" />
    <sdo:userDefinedProperties name="WSRRencoding" value="DEFAULT" />
  </Entity>
  <Entity>
    <NS1:type
      xmlns:NS1="http://www.w3.org/2001/XMLSchema-instance">
      sdo:GenericObject
    </NS1:type>
    <bsrURI>b68952b6-8d68-4840.8e5e.a3716da35e2e</bsrURI>
    <name>ConceptA1</name>
    <namespace>http://www.examples.com/ConceptA1</namespace>
    <version>1.0</version>
    <description>The original concept</description>
    <owner>UNAUTHENTICATED</owner>
    <lastModified>1229030287593</lastModified>
    <creationTimestamp>1227173773250</creationTimestamp>
    <lastModifiedBy>UNAUTHENTICATED</lastModifiedBy>
    <primaryType></primaryType>
    <NS2:classificationURIs
      xmlns:NS2="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
      http://www.ibm.com/xmlns/prod/serviceregistry/6/0/governance/
        DefaultLifecycle#Deprecate
    </NS2:classificationURIs>
    <NS3:userDefinedRelationships
      xmlns:NS3="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS3:userDefinedRelationships>
    <NS4:userDefinedProperties
      xmlns:NS4="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS4:userDefinedProperties>
    <NS5:userDefinedProperties
      xmlns:NS5="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS5:userDefinedProperties>
    <NS6:userDefinedProperties
      xmlns:NS6="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    </NS6:userDefinedProperties>
  </Entity>
</ServiceRegistry>
```

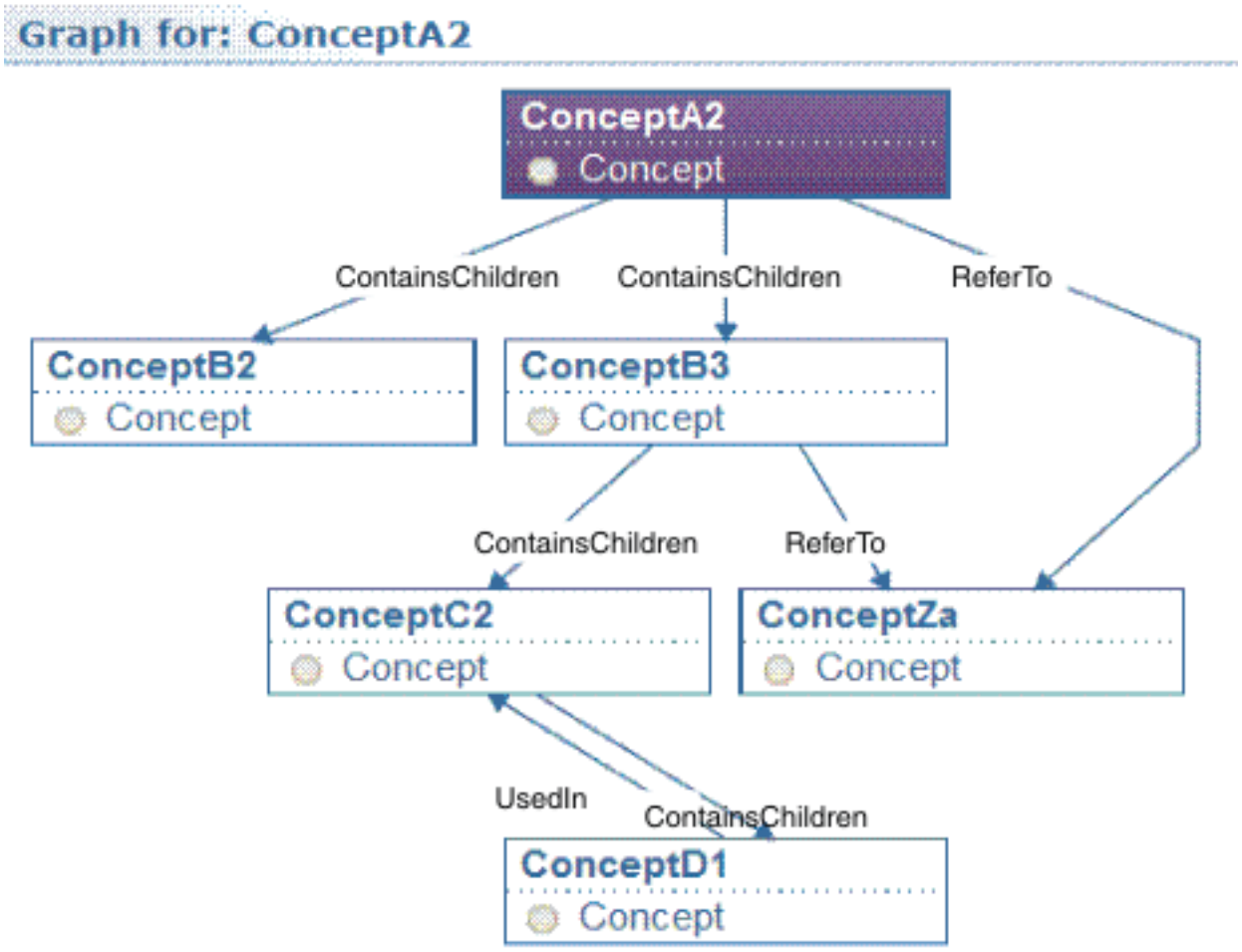
The following ESQL shows how to retrieve the values from the ServiceRegistry message tree in the LocalEnvironment when the `Depth Policy` property is set to `Return matched showing immediate relationships (For compatibility only)`. Note that in this case it is necessary to use the namespace qualifier.

```
DECLARE ns1 NAMESPACE 'http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo';
DECLARE c1 CHARACTER;

-- Following sets c1 to "ConceptA1" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].name;
-- Following sets c1 to "2.0" by indexing the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].version;
-- Following sets c1 to "property1" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].name;
-- Following sets c1 to "value1 for v2" by indexing the first
--   userDefinedProperties within the first entity
SET c1 = InputLocalEnvironment.ServiceRegistry.Entity[1].ns1:
userDefinedProperties[1].value;
```

RegistryLookup node output: example 2

Example showing the structure of RegistryLookup node output for all possible values of the `Depth Policy` property for a query on a concept entity that has a number of user relationships to other concept entities.

This example shows the ServiceRegistry message trees that are stored in the LocalEnvironment when the Concepts shown in the following WebSphere Service Registry and Repository graph are retrieved. The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message trees have some elements replaced by `...` to emphasis the structure of the tree. Likewise, the bsrURIs have been truncated.

The following shows the message trees for each possible value of the `Depth Policy` property:

- `Return matched only (Depth = 0)`

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
```

- `Return matched showing immediate relationships (For compatibility only)`. The entities contain elements showing the details of relationships, but only provide a list of the bsrURIs for the related child entities.

  This value of the `Depth Policy` property is deprecated, so you should use of the other options. The output tree structure produced when using this value is not compatible with those from the other values for the `Depth Policy` property. In particular, note the namespace qualifications.

```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    ns1:userDefinedRelationships
      name    = ContainsChildren
```

```
      targets = b2f73637f6e8 b3de6036e1af
    ns1:userDefinedRelationships
      name    = ReferTo
      targets = zac084d6b804
```

- `Return matched plus immediate related entities (Depth = 1)`. The entities contain elements showing the details of relationships, and the details of the related child entities.

```
ServiceRegistry
  Entity
    type   = GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    userDefinedRelationships
      name = ContainsChildren
      targetEntities
        Entity
          bsrURI = b2f73637f6e8
          name   = ConceptB2
          ...
        Entity
          bsrURI = b3de6036e1af
          name   = ConceptB3
          ...
          userDefinedRelationships
            name    = ContainsChildren
            targets = c26e43ac45a
          userDefinedRelationships
            name    = ReferTo
            targets = zac084d6b804
    userDefinedRelationships
      name = ReferTo
      targetEntities
        Entity
          bsrURI = zac084d6b804
          name   = ConceptZa
          ...
```

- `Return matched plus all related entities (Depth = -1)`. The entities contain elements showing the details of relationships, and the details of the all related child entities. ConceptD1 uses an EntityRef element to refer to its ancestor ConceptC2. ConceptZa appears twice in the tree as it is referenced by both ConceptA2 and ConceptB3.

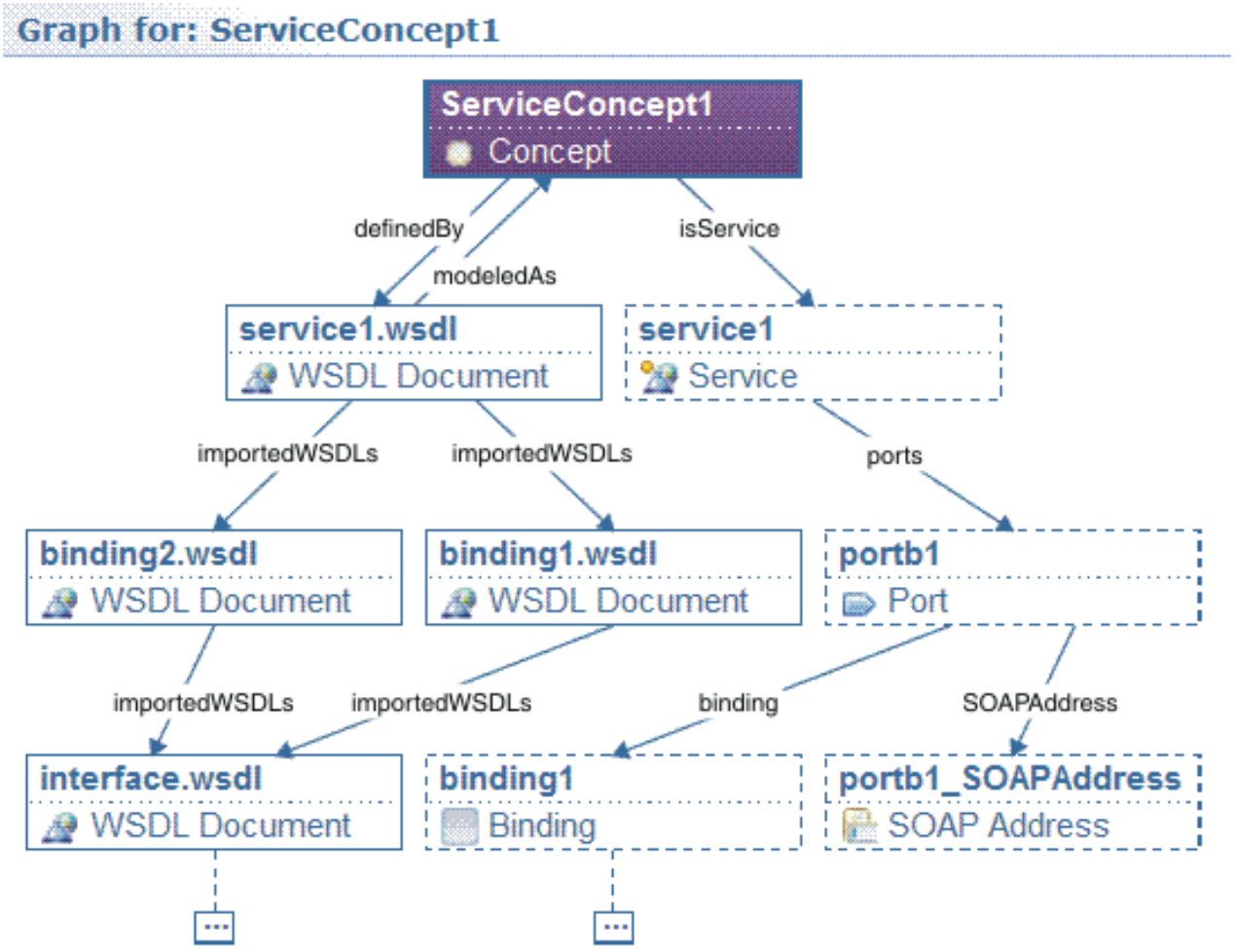
```
ServiceRegistry
  Entity
    type   = sdo:GenericObject
    bsrURI = a2e62137a605
    name   = ConceptA2
    ...
    userDefinedRelationships
      name = ContainsChildren
      targetEntities
        Entity
          bsrURI = b2f73637f6e8
          name   = ConceptB2
          ...
        Entity
          bsrURI = b3de6036e1af
          name   = ConceptB3
          ...
          userDefinedRelationships
            name = ContainsChildren
            targetEntities
              Entity
                bsrURI = c26e43ac45a
                name   = ConceptC2
                ...
                userDefinedRelationships
                  name = ContainsChildren
                  targetEntities
                    Entity
                      bsrURI = d16e43ac763
                      name   = ConceptD1
                      ...
                      userDefinedRelationships
                        name = UsedIn
                        targetEntities
```

```
                          EntityRef
                            bsrURI = c26e43ac45a
                            name   = ConceptC2
          userDefinedRelationships
            name = ReferTo
            targetEntities
              Entity
                bsrURI = zac084d6b804
                name   = ConceptZa
                ...
    userDefinedRelationships
      name = ReferTo
      targetEntities
        Entity
          bsrURI = zac084d6b804
          name   = ConceptZa
          ...
```

RegistryLookup node output: example 3

Example showing the structure of RegistryLookup node output for a query on an entity having metadata relationships and user-defined relationships, using a `Depth Policy` property value of `Return matched plus all related entities (Depth = -1)` .

This example shows the ServiceRegistry message tree that is stored in the LocalEnvironment when the relationships shown in the following WebSphere Service Registry and Repository graph are retrieved using a `Depth Policy` value of `Return matched plus all related entities (Depth = -1)`. The graph has been annotated with the relationship names to clarify the elements in the message tree.



The following ServiceRegistry message tree has some elements replaced by `...` to emphasis the structure of the tree.

```
ServiceRegistry
  Entity
    name = ServiceConcept1
    ...
    userDefinedRelationships
      name = modeledAs
      targetEntities
        Entity
          name = service1.wsdl
          ...
          userDefinedRelationships
            name = definedBy
            targetEntities
              EntityRef
                bsrURI = c26e43ac45a
                name = ServiceConcept1
          metaRelationships
            name = importedWSDLs
            targetEntities
              Entity
                name = binding1.wsdl
                ...
              Entity
                name = binding2.wsdl
                ...
    userDefinedRelationships
      name = isService
      targetEntities
        Entity
          name = service1
          ...
          metaRelationships
            name = ports
            targetEntities
              Entity
                name = portb1
                ...
                metaRelationships
                  name = binding
                  targetEntities
                    Entity
                      name = binding1
                      ...
                metaRelationships
                  name = SOAPAddress
                  targetEntities
                    Entity
                      name = portb1_SOAPAddress
                      ...
```

Connecting to external REST APIs

IBM App Connect Enterprise provides a set of REST nodes, which enable you to interact either synchronously or asynchronously with external REST APIs.

About this task

The following REST nodes are provided with IBM App Connect Enterprise:

- node
- node
- node

An AppConnectRESTRequest node is also provided, which you can use to interact with an IBM App Connect REST API. For more information about using this node, see “Connecting to an App Connect REST API” on page 1047.

You can use a RESTRequest node in a message flow to issue synchronous requests to external REST APIs. The RESTRequest node issues a request and then waits for a response from the REST API. Alternatively, you can use a pair of RESTAsyncRequest and RESTAsyncResponse nodes to interact with a REST API

asynchronously. The RESTAsyncRequest node sends a REST request, and then returns control to the flow without waiting for a response. The RESTAsyncResponse node, which can be in either the same flow or a separate flow, handles the response from the REST API. A separate thread processes the response, and that thread starts a new transaction to handle the response.

You might want to use asynchronous request-response processing when interacting with a backend server that has a high latency. Using synchronous REST requests can result in multiple outstanding requests that require large numbers of threads to make enough simultaneous connections to the backend server, whereas asynchronous processing allows the response to be decoupled from the request.

For more information about using a RESTRequest node in a message flow, see node. For more information about using the asynchronous REST nodes, see node and node.

Procedure

Complete the steps in one the following tasks to configure interactions with external REST APIs by using REST nodes in a message flow:

- "Calling REST APIs by using the RESTRequest node" on page 1033
- "Interacting asynchronously with REST APIs by using the RESTAsyncRequest and RESTAsyncResponse nodes" on page 1035

Calling REST APIs by using the RESTRequest node

IBM App Connect Enterprise provides a RESTRequest node, which you can use to interact with external REST APIs.

About this task

You can use a RESTRequest node in a message flow to issue synchronous requests to external REST APIs. The RESTRequest node uses an imported Swagger document or an imported OpenAPI 3.0 document in either JSON or YAML format. You can import the document from the toolkit workspace, your file system, or from a URL.

If you created a REST API from scratch in the IBM App Connect Enterprise Toolkit, a Swagger document or an OpenAPI 3.0 document was created. You can import the document and use it with the RESTRequest node.

For more information about creating REST APIs, see "Creating a REST API" on page 2019. For more information about using a RESTRequest node, see node.

Procedure

Import a Swagger document or an OpenAPI 3.0 document and configure a RESTRequest node by completing the following steps:

1. Create a message flow to include your RESTRequest node.

   For more information about creating a message flow, see "Creating a message flow" on page 574.
2. Populate the **REST APIs Catalog**.

   Each project in the Application Development view has a **REST APIs Catalog** folder that contains a list of all the REST APIs that are available in the project for use with RESTRequest and RESTAsyncRequest

nodes. Each REST API folder in the **REST APIs Catalog** contains a list of all the available operations in the REST API. Populate the **REST APIs Catalog** by completing the following steps:

a) Right-click a project or an application in the Application Development view and select **Import**. Alternatively, select **File** > **Import**.

b) Select **RESTAPIs** > **RestAPI definition** to open the **Import a RESTAPI definition** dialog.

c) Select a project from the **Project** menu or click **New** to create a new project.

d) Select **Select an OpenAPI document from the workspace or from a file system** or **Retrieve an OpenAPI document from a URL using HTTP or HTTPS**.

e) Click **Next** to open the **Import a RESTAPI definition** dialog.

f) Depending on the choice you made at substep d, you are prompted to select a file from a file system or a project in the workspace, or specify a URL.

g) Click **Next** to review the details of the RESTAPI that you want to import.

h) If the details of the RESTAPI are correct, click **Next**

The Swagger document or OpenAPI 3.0 document is copied into the **REST APIs Catalog** and the **Other resources** folder for the project that contains the RESTRequest node, and is visible in the Application Development view.

3. Configure the RESTRequest node to use an imported Swagger document or an imported OpenAPI 3.0 document that contains the operation that you want to invoke, by using one of the following methods:

   - Drag an operation from the **REST APIs Catalog** onto the message flow canvas or onto a connection between two existing nodes in the message flow. A RESTRequest node is created automatically and configured to use the selected operation. If the operation is dropped onto a connection between two existing nodes, the RESTRequest is inserted into the flow between the two existing nodes and configured to use the selected operation.
   - Drag a RESTRequest node from the palette onto the message flow canvas or onto a connection between two existing nodes in your message flow. If you drag the RESTRequest node onto the canvas, a configuration wizard opens. If you drag a RESTRequest node from the palette onto a connection between two existing nodes in your message flow, the RESTRequest node is inserted into the flow between the two existing nodes, and the configuration wizard opens.

     You must then use the configuration wizard opens to choose the REST API and the operation that you want to invoke. You can select an operation from the set of operations in the imported Swagger document or imported OpenAPI 3.0 document. You can also specify parameter values as node properties, by using XPath or ESQL expressions to select data from the input message tree.

     a. Use the **Invoke an operation in a REST API** page in the wizard to specify the location of the Swagger document or OpenAPI 3.0 document that contains the operation that you want to use:

        – **Select a Swagger document or an OpenAPI 3.0 document from a referenced project or from a file system**

          Select this option to display a window in which you can specify either a location in your file system, or select a Swagger document or an OpenAPI 3.0 document from a referenced

project. If you select a Swagger document or an OpenAPI 3.0 document from your file system, it is copied into the project that contains your RESTRequest node.

If you want to use a Swagger document or an OpenAPI 3.0 document from a shared library that is not referenced, you must reference the shared library, and then add the RESTRequest node.

- **Retrieve a Swagger document or an OpenAPI 3.0 document from a URL using HTTP or HTTPS**

  Select this option to display a window in which you can specify a URL to a Swagger document or an OpenAPI 3.0 document. This Swagger document or OpenAPI 3.0 document is downloaded and stored in the project that contains the RESTRequest node.

- **Configure the REST request node manually**

  Select this option to close the wizard without changing the RESTRequest node. You can then optionally drag an operation from the **REST APIs Catalog** in the Navigation onto the RESTRequest node in the message flow, and it is configured automatically to use that operation.

b. If you selected either **Select a Swagger document or an OpenAPI 3.0 document from a referenced project or from a file system** or **Retrieve a Swagger document or an OpenAPI 3.0 document from a URL using HTTP or HTTPS** in the previous step, a window is displayed in which you can select an operation to invoke in the REST API. Select the required operation and click **Finish**. The RESTRequest node is automatically configured to use the selected operation.

4. Configure the remaining properties of your RESTRequest node.

   For more information, see node.

5. Configure the security credentials for interacting with the REST API, by using the command command.

   For more information, see “Configuring security credentials for connecting to a REST API” on page 1037.

Interacting asynchronously with REST APIs by using the RESTAsyncRequest and RESTAsyncResponse nodes

You can make REST requests and receive responses asynchronously by using RESTAsyncRequest and RESTAsyncResponse nodes in your message flow.

About this task

You might want to use asynchronous request-response processing with a backend server that has a high latency. Synchronous processing can result in many outstanding requests that require large numbers of threads to make simultaneous connections to the backend server. Asynchronous processing allows the response to be decoupled from the request.

You can use a pair of RESTAsyncRequest and RESTAsyncResponse nodes to interact with a REST API asynchronously. The paired nodes correlate responses against the original requests by using a unique identifier, which is specified on both nodes. Only one pair of nodes in an execution group can have the same unique identifier.

The RESTAsyncRequest node sends a REST request, and then returns control to the flow without waiting for a response. This action frees the request thread to handle further requests, while the response is handled by the paired RESTAsyncResponse node on a different thread and in a new transaction. This asynchronous functionality enables multiple outbound requests to be made almost in parallel because the outbound request is not blocked waiting for the response. The RESTAsyncResponse node can be in either the same flow or a separate flow, but it must be in the same integration server as its paired request node.

You can specify a timeout interval, so that if the whole request-response operation takes longer than the specified duration, the request is propagated to the Failure terminal on one of the paired nodes. The timeout interval applies to the interval that starts with the request being sent by the RESTAsyncRequest node, and ends with the response being completely received by the RESTAsyncResponse node.

For each request that the RESTAsyncRequest node processes, it uses a connection to send the request, and passes the connection to the paired RESTAsyncResponse node for the response. If the timeout interval is specified, the socket is closed if the interval expires. This closure ensures that a request gets only the correct response, and any response data for a request that is timed out is discarded.

The request-response processing is split between the RESTAsyncRequest node and the RESTAsyncResponse node. If a timeout occurs during the request phase of processing, the request and an exception are propagated to the Failure terminal of the RESTAsyncRequest node. Similarly, if a timeout occurs during the response phase of processing, an exception is propagated to the Failure terminal of the RESTAsyncResponse node. In most cases, the timeout would occur when waiting for a server response, in which case the Failure terminal of the RESTAsyncResponse node is used. In rare cases a timeout might occur when the request node sends data to the server. In this case, the Failure terminal of the RESTAsyncRequest node is used.

Procedure

Import a Swagger document or an OpenAPI 3.0 document and configure a RESTAsyncRequest node by completing the following steps:

1. Create a message flow to include your RESTAsyncRequest node.

   For more information about creating a message flow, see “Creating a message flow” on page 574.

2. Optional: If you intend to Configure the REST request node manually at step 3, you must populate the **REST APIs Catalog**.

   Each project in the Application Development view has a **REST APIs Catalog** folder that contains a list of all the REST APIs that are available in the project for use with RESTRequest and RESTAsyncRequest nodes. Each REST API folder in the **REST APIs Catalog** contains a list of all the available operations in the REST API. Populate the **REST APIs Catalog** by completing the following steps:

   a) Right-click a project or an application in the Application Development view and select **Import**. Alternatively, select **File** > **Import**.

   b) Select **RESTAPIs** > **RestAPI definition** to open the **Import a RESTAPI definition** dialog.

   c) Select a project from the **Project** menu or click **New** to create a new project.

   d) Select **Select an OpenAPI document from the workspace or from a file system** or **Retrieve an OpenAPI document from a URL using HTTP or HTTPS**.

   e) Click **Next** to open the **Import a RESTAPI definition** dialog.

   f) Depending on the choice you made at substep d, you are prompted to select a file from a file system or a project in the workspace, or specify a URL.

   g) Click **Next** to review the details of the RESTAPI that you want to import.

   h) If the details of the RESTAPI are correct, click **Next**

   The Swagger document or OpenAPI 3.0 document is copied into the **REST APIs Catalog** and the **Other resources** folder for the project that contains the RESTRequest node, and is visible in the Application Development view.

3. Drag a RESTAsyncRequest node from the palette onto the message flow canvas, and configure it to use an imported Swagger document or an imported OpenAPI 3.0 document that contains the operation that you want to invoke.

   Alternatively, you can drag a RESTAsyncRequest node from the palette onto a connection between two existing nodes in your message flow. The RESTAsyncRequest node is inserted into the flow between the two existing nodes.

   When you place the node onto the canvas, a wizard guides you through choosing the REST API and the operation that you want to invoke. You can select an operation from the set of operations in the

imported Swagger document or the imported OpenAPI 3.0 document, and specify parameter values as node properties, by using XPath or ESQL expressions to select data from the input message tree.

a) Use the **Invoke an operation in a REST API** page in the wizard to specify the location of the Swagger document or OpenAPI 3.0 document that contains the operation that you want to use:

- **Select a Swagger document or an OpenAPI 3.0 document from a referenced project or from a file system**

  Select this option to display a window in which you can specify either a location in your file system, or select a Swagger document or OpenAPI 3.0 document from a referenced project. If you select a Swagger document or an OpenAPI 3.0 document from your file system, it is copied into the project that contains your RESTAsyncRequest node.

  If you want to use a Swagger document or an OpenAPI 3.0 document from a shared library that is not referenced, you must reference the shared library first, and then add the RESTAsyncRequest node.

- **Retrieve a Swagger document or an OpenAPI 3.0 document from a URL using HTTP or HTTPS**

  Select this option to display a window in which you can specify a URL to a Swagger document or an OpenAPI 3.0 document. This Swagger document or OpenAPI 3.0 document is downloaded and stored in the project that contains the RESTAsyncRequest node.

- **Configure the REST request node manually**

  Select this option to close the wizard without changing the RESTAsyncRequest node. You can then optionally drag an operation from the **REST APIs Catalog** in the Navigation onto the RESTAsyncRequest node in the message flow, and it is configured automatically to use that operation.

b) If you selected either **Select a Swagger document or an OpenAPI 3.0 document from a referenced project or from a file system** or **Retrieve a Swagger document or an OpenAPI 3.0 document from a URL using HTTP or HTTPS** in the previous step, a window is displayed in which you can select an operation to invoke in the REST API. Select the required operation and click **Finish**. The RESTAsyncRequest node is automatically configured to use the selected operation.

4. Configure the remaining properties of your RESTAsyncRequest node.

   For more information, see node.

5. Add a RESTAsyncResponse node to handle the response to your REST request. This node can be in either the same flow or a separate flow, but it must be in the same integration server.

   For more information, see node.

6. Configure the security credentials for interacting with the REST API, by using the command command.

   For more information, see “Configuring security credentials for connecting to a REST API” on page 1037.

7. You can share data between the flow containing the RESTAsyncRequest node and the flow containing the RESTAsyncResponse node, by storing and retrieving data in the UserContext folder in the Local Environment. You can store data in the request flow under `OutputLocalEnvironment.Destination.REST.Request.UserContext` and you can retrieve data in the response flow from `LocalEnvironment.REST.Response.UserContext`. For examples, see “Using local environment variables with REST nodes” on page 1039.

Configuring security credentials for connecting to a REST API

Create a security identity by using the **`mqsisetdbparms`** command, and configure a REST request node in a message flow to use that identity for connecting to a secured REST API.

About this task

Follow these steps to configure a connection to a secured REST API:

Procedure

1. Use the **mqsisetdbparms** command to associate a user name, password, and API key with a connection to an external REST API.

   You can specify these values in any one of the following combinations:

   - User ID, password, and API key
   - User ID and password
   - API key only

   You can specify the security credentials by setting the following parameters:

   -n *rest::securityIdentity*
   The name of the security identity that is used to authenticate a connection to a REST API, where *securityIdentity* is the value of the `Security Identity` property in the RESTRequest or RESTAsyncRequest node.

   -u *UserId*
   The user ID to be used for connecting to the REST API.

   -p *Password*
   The password to be used for connecting to the REST API.

   -k *API key*
   The API key to be used for connecting to the REST API.

   The following example shows how to specify a user ID, password, and API key:

   ```
   mqsisetdbparms -w workDir -n rest::myRESTSecurityIdentity -u myRESTUserID -p myRESTPassword -
   k myRESTAPIKey
   ```

   Alternatively, you might choose to specify only the API key, as shown in the following example:

   ```
   mqsisetdbparms -w workDir -n rest::myRESTSecurityIdentity -k myRESTAPIKey
   ```

   For more information about associating security credentials with resources, see command.

   If the configured REST API operation contains multiple "securityDefinitions" each of which requires a different set of credentials, you must create a separate "Security Identity" for each "securityDefinition". This can be achieved only by using LocalEnvironment overrides for "Security Identity" as detailed in “Using local environment variables with REST nodes” on page 1039.

2. In your message flow, specify the name of the security identity that you configured in step 1 (in this example, `myRESTSecurityIdentity`) as the value in the `Security identity` property in the RESTRequest or RESTAsyncRequest node.

   This security identity is used when you connect to the REST API from IBM App Connect Enterprise.

   In order for the REST request nodes to authenticate to a REST API by using a security identity created with the **mqsisetdbparms** command, the REST request nodes require additional information about how to apply that security identity to the REST API. The REST request nodes use the specified security identity only if the operation or API being invoked specifies one or more security requirements in the Swagger document or OpenAPI 3.0 document.

   Security requirements can be specified at the API level, or for a specific operation, by adding Security Requirement Objects to the **security** property of the Swagger Object, the OpenAPI 3.0 object, or the relevant Operation Object. Each Security Requirement Object must refer to a Security Scheme Object in the Security Definitions Object, which is defined through the **securityDefinitions** property of the Swagger object or OpenAPI 3.0 object. IBM App Connect Enterprise supports only Security

Scheme Objects that have a type of `basic` or `apiKey`. Security Scheme Objects with a type of `oauth2` are not supported.

For more information about Swagger, see the following topics:

- Swagger
- Swagger RESTful API Documentation Specification Version 2.0
- “Restrictions on Swagger documents” on page 2018

For more information about OpenAPI 3.0, see the following topics:

- “OpenAPI 3.0” on page 2018
- https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.0.md
- “Restrictions on OpenAPI 3.0 documents” on page 2019

What to do next

You can use the **mqsireportdbparms** command to show information about the security identities that are being used for connecting to a REST API. For example:

- Enter the following command to show the security identity for HTTP Basic Authentication and the API key, for use with a REST request node:

```
mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuthAndApiKey
```

This command returns output similar to the following example:

```
$ mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuthAndApiKey
BIP8180I: The resource name 'rest::myBasicAuthAndApiKey' has userID 'myUserID'.
BIP8214I: The resource name 'rest::myBasicAuthAndApiKey' has API key
 'C664C588-885A-4F07-9390-9CD7A4F8A89F'.
```

- Enter the following command to validate the password for a single security identity for HTTP Basic Authentication, for use with a REST request node:

```
mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuth -u myUserID -p password
```

This command returns output similar to the following example:

```
$ mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuth -u myUserID -p password
BIP8180I: The resource name 'rest::myBasicAuth' has userID 'myUserID'.
BIP8201I: The password you entered, 'password' for resource 'rest::myBasicAuth' and userId
 'myUserID' is correct.


BIP8071I: Successful command completion.
```

For more information about the security credentials that were set on the integration server, see command.

Using local environment variables with REST nodes

The REST request nodes support a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node properties.

The following table shows elements in the **LocalEnvironment.Destination.REST.Request** message tree, which can be used to override properties in the RESTRequest and RESTAsyncRequest nodes. Any properties that are set by the local environment variables will apply to both the RESTRequest and RESTAsyncRequest nodes.

The table includes examples of how to set the values by using ESQL; however, you can also set them by using transformation nodes, such as the Mapping node. To access the LocalEnvironment from a Mapping node, see Customizing a message map to include a message assembly component.

Table 25. Input local environment properties

Element name	Type	Description
Operation	string	The value is the name of the operation to invoke in the REST API. This environment variable overrides the **Operation** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.Operation = 'updateCustomerByID';`

Parameters.*parameter_name*	string	Specify values to use for the parameters of the operation. The values specified in the LocalEnvironment override any literal values or XPath/ESQL expressions specified in the **Parameters** table on the node. For example:
		`SET OutputLocalEnvironment.Destination.REST.Request.Parameters.max = 10;`
		`SET OutputLocalEnvironment.Destination.REST.Request.Parameters.filter = 'Fred Bloggs';`
		Note: If the values are XPath/ESQL expressions, the values are passed to the remote REST API as string literals (and not evaluated as XPath/ESQL expressions).
		To define the REST parameters and set their values in a message map, use the **Add User Defined** function to add the `parameter_name` element, and then provide a mapping to set its value. For more information, see Configuring the local environment tree Variables folder by using the Add User Defined function.

ContentType	string	The value is the value of the Content-Type header to send in the request to the REST API. This environment variable overrides the **Content-Type** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.ContentType = 'application/vnd.ibm-demo+json';`
Accept	string	The value is the value of the Accept header to send in the request to the REST API. This environment variable overrides the **Accept** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.Accept = 'application/json';`
SecurityIdentity	string	The value must be the name of a security identity defined with the **mqsisetdbparms** command. Do not include the "rest::" prefix. This environment variable overrides the **Security identity** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.SecurityIdentity = 'myApiKey';`

Table 25. Input local environment properties (continued)

Element name	Type	Description
UserID	string	If this environment variable is specified, it overrides all the values that are specified by the `LocalEnvironment.Destination.REST.Request.SecurityIdentity` override and the **Security identity** property on the node. If the `LocalEnvironment.Destination.REST.Request.UserID` environment variable is specified, the `LocalEnvironment.Destination.REST.Request.Password` environment variable is also required.
Password	string	If this environment variable is specified, it overrides all the values that are specified by the `LocalEnvironment.Destination.REST.Request.SecurityIdentity` override and the **Security identity** property on the node. If the `LocalEnvironment.Destination.REST.Request.Password` environment variable is specified, the `LocalEnvironment.Destination.REST.Request.UserID` environment variable is also required.
APIKey	string	If this environment variable is specified, it overrides all the values that are specified by the `LocalEnvironment.Destination.REST.Request.SecurityIdentity` override and the **Security identity** property on the node. The `LocalEnvironment.Destination.REST.Request.APIKey` environment variable can be specified either alone or in conjunction with the `LocalEnvironment.Destination.REST.Request.UserID` and `LocalEnvironment.Destination.REST.Request.Password` environment variables.
Timeout	integer	The time (in seconds) that the node waits for the REST API to process the operation. This environment variable overrides the **Request timeout (sec)** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.Timeout = 300;`
BaseURL	string	This environment variable overrides the **Base URL override** property on the node, and the base URL specified in the Swagger document containing the definitions of the REST API. For example: `SET OutputLocalEnvironment.Destination.REST.Request.BaseURL = 'https://my-prod-server.ibm.com/customerdb/v1';`
ProxyURL	string	This environment variable overrides the **HTTP(S) proxy location** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.ProxyURL = 'http://my-proxy-server.ibm.com';`

Table 25. Input local environment properties (continued)

Element name	Type	Description
FollowRedirection	boolean	This environment variable overrides the **Follow HTTP(S) redirection** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.FollowRedirection = TRUE;`
KeepAlive	boolean	This environment variable overrides the **Enable HTTP 1/1 keep-alive** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.KeepAlive = FALSE;`
Compression	string	This environment variable overrides the **Compression** property on the node. Valid values are: • gzip • zlib-deflate • raw-deflate For example: `SET OutputLocalEnvironment.Destination.REST.Request.Compression = 'zlib-deflate';`
Protocol	string	This environment variable overrides the **Protocol** property on the node. Valid values are: • TLS • TLSv1 • TLSv1.2 • TLSv1.3 • SSL_TLS • SSL_TLSv2 For example: `SET OutputLocalEnvironment.Destination.REST.Request.Protocol = 'TLSv1.2';`
AllowedCiphers	string	This environment variable overrides the **Allowed SSL Ciphers** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.AllowedCiphers = 'SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA';`
HostnameChecking	boolean	This environment variable overrides the **Enable SSL certificate hostname checking** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.HostnameChecking = FALSE;`

Table 25. Input local environment properties (continued)

Element name	Type	Description
KeyAlias	string	This environment variable overrides the **SSL client authentication key alias** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.KeyAlias = 'myClientCertificate';`
EnableCRLCheck	boolean	This environment variable overrides the **Enable certificate revocation list checking** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.EnableCRLCheck = TRUE;`
AcceptCompressedResponses	string	This environment variable overrides the **Accept compressed responses by default** property on the node. For example: `SET OutputLocalEnvironment.Destination.REST.Request.AcceptCompressedResponses = TRUE;`
ServicePrincipalName	string	Specifies the Service Principal Name (SPN) to use when the integration node negotiates the Kerberos security protocol. For example: `SET OutputLocalEnvironment.Destination.REST.Request.ServicePrincipalName = 'HTTP/iib.iibservice2.com:7800';`
SecuritySchemes.*security_scheme*.SecurityIdentity	string	Specifies the name of a security identity to use for the specified security scheme. The specified security identity is used only for the specified security scheme. If this environment variable is specified, it overrides the `LocalEnvironment.Destination.REST.Request.SecurityIdentity` override and the **Security identity** property on the node. The name of the security scheme corresponds to the name of a Security Scheme Object in a Security Definitions Object in the Swagger document containing the definitions of the REST API. The value must be the name of a security identity defined with the **mqsisetdbparms** command. Do not include the "rest::" prefix. For example: `SET OutputLocalEnvironment.Destination.REST.Request.SecuritySchemes.petstore_aut = 'myIdentity3'`

Table 25. Input local environment properties (continued)

Element name	Type	Description
SecuritySchemes.*security_scheme*.UserID	string	Specify the user ID to use for the specified security scheme when using HTTP Basic Authentication. The specified user ID is used only for the specified security scheme. If this environment variable is specified, it overrides the user ID specified in a security identity using either the `LocalEnvironment.Destination.REST.Request.SecuritySchemes.secur` or `LocalEnvironment.Destination.REST.Request.SecurityIdentity` overrides, and also the **Security identity** property on the node. The name of the security scheme corresponds to the name of a Security Scheme Object in a Security Definitions Object in the Swagger document containing the definitions of the REST API. For example: `SET OutputLocalEnvironment.Destination.REST.Request.SecuritySchemes.security_sch`
SecuritySchemes.*security_scheme*.Password	string	Specify the password to use for the specified security scheme when using HTTP Basic Authentication. The specified password is used only for the specified security scheme. If this environment variable is specified, it overrides the password specified in a security identity using either the `LocalEnvironment.Destination.REST.Request.SecuritySchemes.secur` or `LocalEnvironment.Destination.REST.Request.SecurityIdentity` overrides, and also the **Security identity** property on the node. The name of the security scheme corresponds to the name of a Security Scheme Object in a Security Definitions Object in the Swagger document containing the definitions of the REST API. For example: `SET OutputLocalEnvironment.Destination.REST.Request.SecuritySchemes.petstore_aut`
SecuritySchemes.*security_scheme*.APIKey	string	Specify the API key to use for the specified security scheme when using HTTP Basic Authentication. The specified API key is used only for the specified security scheme. If this environment variable is specified, it overrides the API key specified in a security identity using either the `LocalEnvironment.Destination.REST.Request.SecuritySchemes.secur` or `LocalEnvironment.Destination.REST.Request.SecurityIdentity` overrides, and also the **Security identity** property on the node. The name of the security scheme corresponds to the name of a Security Scheme Object in a Security Definitions Object in the Swagger document containing the definitions of the REST API. For example: `SET OutputLocalEnvironment.Destination.REST.Request.SecuritySchemes.petstore_aut`

Table 25. Input local environment properties (continued)

Element name	Type	Description

UserContext	mixed content	Specify context data to be stored by the RESTAsyncRequest node. The data can later be accessed by the RESTAsyncResponse node for the matching request.
		
		You can use the user context to specify data that will be passed from the RESTAsyncRequest node to the RESTAsyncResponse node. To store user context data, prior to the RESTAsyncRequest node, either set a value in the `OutputLocalEnvironment.Destination.REST.Request.UserContext` environment variable or create and populate child nodes below it.
		
		For example, to set the values by using an ESQL expression:
		
		`SET OutputLocalEnvironment.Destination.REST.Request.UserContext = 'Some important information';`
		
		`SET OutputLocalEnvironment.Destination.REST.Request.UserContext.Name.FirstName = 'Fred';`
		`SET OutputLocalEnvironment.Destination.REST.Request.UserContext.Name.Surname = 'Bloggs';`
		
		`SET OutputLocalEnvironment.Destination.REST.Request.UserContext.MyData.Binary = CAST('A piece of data' AS BLOB CCSID 1208);`
		
		You can use a Mapping node to set a value, by mapping to the mixed child element `OutputLocalEnvironment.Destination.REST.Request.UserContext.mix` To use a Mapping node to create sub-elements, use user-defined elements or cast `xsd:any` and map to them. For more information, see node.
		
		To access the user context data after a RESTAsyncResponse node, read the value or children of the `LocalEnvironment.REST.Response.UserContext` environment variable.

QueryStringDoNotPercentEncodeChars	String	Specifies those characters in the query string that are not to be percent encoded while the URI is built. For example, to specify that the comma and greater than symbols are not to be percent encoded: `SET OutputLocalEnvironment.Destination.REST.Request.QueryStringDoNotPercentEncod = ',>';` The default set of characters not to be percent encoded is asterisk (*), minus (-), period (.), and underscore (_).

The following table shows the data that is written by the RESTRequest node to the LocalEnvironment when propagating an output message:

Table 26. Output local environment properties generated by the RESTRequest node		
Element	**Type**	**Description**
LocalEnvironment.WrittenDestination.REST.Method	string	The HTTP method used when invoking the operation in the REST API.
LocalEnvironment.WrittenDestination.REST.URL	string	The URL used when invoking the operation in the REST API.
LocalEnvironment.WrittenDestination.REST.RequestHeaderSize	integer	The size of the request headers sent to the REST API.
LocalEnvironment.WrittenDestination.REST.RequestBodySize	integer	The size of the request body sent to the REST API.
LocalEnvironment.WrittenDestination.REST.StatusCode	integer	The HTTP status code in the response from the REST API.
LocalEnvironment.WrittenDestination.REST.ResponseHeaderSize	integer	The size of the response headers received from the REST API.
LocalEnvironment.WrittenDestination.REST.ResponseBodySize	integer	The size of the response body received from the REST API.
LocalEnvironment.WrittenDestination.REST.TotalRequestTime	integer	The total elapsed time invoking the operation in the REST API.

The following table shows the data that is written by the RESTAsyncRequest node when propagating an output message:

Table 27. Output local environment properties generated by the RESTAsyncRequest node:		
Element	**Type**	**Description**
LocalEnvironment.WrittenDestination.REST.Method	string	The HTTP method used when invoking the operation in the REST API.
LocalEnvironment.WrittenDestination.REST.URL	string	The URL used when invoking the operation in the REST API.
LocalEnvironment.WrittenDestination.REST.RequestHeaderSize	integer	The size of the request headers sent to the REST API.
LocalEnvironment.WrittenDestination.REST.RequestBodySize	integer	The size of the request body sent to the REST API.
LocalEnvironment.WrittenDestination.REST.CorrelationID	blob	The correlation ID used to correlate the request and response between the RESTAsyncRequest and RESTAsyncResponse nodes.

The following table shows the data that is written by the RESTAsyncResponse node when propagating an output message:

Table 28. Output local environment properties generated by the RESTAsyncResponse node:		
Element	**Type**	**Description**
LocalEnvironment.REST.Response.CorrelationID	blob	The correlation ID used to correlate the request and response between the RESTAsyncRequest and RESTAsyncResponse nodes.
LocalEnvironment.REST.Response.StatusCode	integer	The HTTP status code in the response from the REST API.
LocalEnvironment.REST.Response.ResponseHeaderSize	integer	The size of the response headers received from the REST API.
LocalEnvironment.REST.Response.ResponseBodySize	integer	The size of the response body received from the REST API.

Table 28. Output local environment properties generated by the RESTAsyncResponse node: (continued)		
Element	**Type**	**Description**
LocalEnvironment.REST.Response.UserContext	mixed content	The information that was stored in the OutputLocalEnvironment.Destination.REST.Request.UserContext can be retrieved by the response thread.

Connecting to an App Connect REST API

IBM App Connect Enterprise provides an AppConnectRESTRequest node, which enables interaction with an IBM App Connect REST API.

About this task

You can use an AppConnectRESTRequest node in a message flow to issue requests to an IBM App Connect REST API. The AppConnectRESTRequest node uses an imported Swagger document or an imported OpenAPI 3.0 document (in either JSON or YAML format), which defines the REST API and the operations that are available. When you download the Swagger document or the OpenAPI 3.0 document, you can import it from your toolkit workspace and select the operations that you want to invoke. For more information, see node.

Alternatively, you can configure a connection to an App Connect API by using an App Connect REST Request pattern, which is available through the Patterns Explorer. For more information about using patterns, see “Developing integration solutions by using patterns” on page 2056.

Procedure

1. From IBM App Connect, download the Swagger document or the OpenAPI 3.0 document for the App Connect API that you want to use, such as the GoogleSheetsAPI. Import the document into the appropriate application folder in the IBM App Connect Enterprise workspace.

   To import the API document, right click the application and select **Import** > **REST APIs** > **REST API definition** to open the **Import a REST API definition** dialog. Then, pick either **select from workspace** or **retrieve from a URL** to import the document.

   When you have imported it, the Swagger document or OpenAPI 3.0 document is visible in the Application Development view in the **REST APIs Catalog** and the **Other resources** folder of your project that contains your application.
2. Create a message flow to include the AppConnectRESTRequest node.

   For more information about creating a message flow, see “Creating a message flow” on page 574.
3. Drag an AppConnectRESTRequest node from the palette onto the message flow canvas.
4. Configure the AppConnectRESTRequest node by setting the following properties in the **Basic** tab of the node:
   a) In the **Definitions file** field, specify the Swagger document or OpenAPI 3.0 document that contains the definitions of the REST API operations that you want to invoke.

      To select the Swagger document or the OpenAPI 3.0 document, click **Browse**, and then select the required file from the list that is displayed in the REST API Definitions File Selection dialog, and click **OK**. (The file is in `*.json` or `*.yaml` format.)

      The operations that are contained in the selected App Connect API (such as `GoogleSheets.create`) are now listed in the **Operation** field of the AppConnectRESTRequest node.
   b) Select the operation that you want to invoke.
5. The **Request** tab automatically lists any parameters that are required for the selected operation. You can enter a literal value, a location in the message tree, or a location in the Local Environment.
6. Configure the remaining properties of your AppConnectRESTRequest node.

   For more information about these properties, see node.

7. Configure the security credentials for interacting with the REST API, by using the command command.

   For more information, see "Configuring security credentials for connecting to an IBM App Connect REST API" on page 1048.

Configuring security credentials for connecting to an IBM App Connect REST API

Create a security identity by using the **mqsisetdbparms** command, and configure an AppConnectRESTRequest node in a message flow to use that identity for connecting to a secured App Connect REST API.

About this task

Follow these steps to configure a connection to a secured App Connect REST API:

Procedure

1. Use the **mqsisetdbparms** command to associate a user name, password, and API key with a connection to an App Connect REST API.

   You can specify these values in any one of the following combinations:

   - User ID, password, and API key
   - User ID and password
   - API key only

   You can specify the security credentials by setting the following parameters:

   -n *rest::securityIdentity*
   The name of the security identity that is used to authenticate a connection to an App Connect REST API, where *securityIdentity* is the value of the `Security Identity` property in the AppConnectRESTRequest node.

   -u *UserId*
   The user ID to be used for connecting to the App Connect REST API.

   -p *Password*
   The password to be used for connecting to the App Connect REST API.

   -k *API key*
   The API key to be used for connecting to the App Connect REST API.

   The following example shows how to specify a user ID, password, and API key:

   ```
   mqsisetdbparms -w workDir -n rest::myRESTSecurityIdentity -u myRESTUserID -p myRESTPassword -
   k myRESTAPIKey
   ```

   Alternatively, you might choose to specify only the API key, as shown in the following example:

   ```
   mqsisetdbparms -w workDir -n rest::myRESTSecurityIdentity -k myRESTAPIKey
   ```

   For more information about associating security credentials with resources, see command.

2. In your message flow, specify the name of the security identity that you configured in step 1 (in this example, `myRESTSecurityIdentity`) as the value in the `Security identity` property in the AppConnectRESTRequest node.

   This security identity will be used when you connect to the App Connect REST API from IBM App Connect Enterprise.

   In order for the AppConnectRESTRequest request node to authenticate to a REST API using a security identity created with the **mqsisetdbparms** command, the AppConnectRESTRequest node requires additional information about how to apply that security identity to the REST API. The AppConnectRESTRequest node uses the specified security identity only if the operation or API being invoked specifies one or more security requirements in the Swagger document.

What to do next

You can use the **mqsireportdbparms** command to show information about the security identities that are being used for connecting to a REST API. For example:

- Enter the following command to show the security identity for HTTP Basic Authentication and the API key, for use with an AppConnectRESTRequest node:

  ```
  mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuthAndApiKey
  ```

  This command returns output similar to the following example:

  ```
  $ mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuthAndApiKey
  BIP8180I: The resource name 'rest::myBasicAuthAndApiKey' has userID 'myUserID'.
  BIP8214I: The resource name 'rest::myBasicAuthAndApiKey' has API key
   'C664C588-885A-4F07-9390-9CD7A4F8A89F'.
  ```

- Enter the following command to validate the password for a single security identity for HTTP Basic Authentication, for use with an AppConnectRESTRequest request node:

  ```
  mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuth -u myUserID -p password
  ```

  This command returns output similar to the following example:

  ```
  $ mqsireportdbparms -w c:\workdir\ACEServ1 -n rest::myBasicAuth -u myUserID -p password
  BIP8180I: The resource name 'rest::myBasicAuth' has userID 'myUserID'.
  BIP8201I: The password you entered, 'password' for resource 'rest::myBasicAuth' and userId
   'myUserID' is correct.

  BIP8071I: Successful command completion.
  ```

For more information about the security credentials that have been set on the integration server, see command.

Connecting to Enterprise Information Systems

Use WebSphere Adapters to communicate with Enterprise Information Systems (EIS) such as SAP, Siebel, PeopleSoft, and JD Edwards.

About this task

This section contains the following concept information:

- “WebSphere Broker Adapters Transport” on page 1050
- “WebSphere Adapters nodes” on page 1050
- “Overview of WebSphere Adapter for SAP Software” on page 1052
- “Overview of WebSphere Adapter for Siebel Business Applications ” on page 1092
- “Overview of WebSphere Adapter for PeopleSoft Enterprise” on page 1098
- “Overview of WebSphere Adapter for JD Edwards EnterpriseOne” on page 1102

This section contains the following tasks:

- “Developing message flows that use WebSphere Adapters” on page 1106
- “Preparing your system to use WebSphere Adapters nodes” on page 1106
- “Preparing the environment for WebSphere Adapters nodes” on page 209
- “Activating IBM License Metric Tool for WebSphere Adapters” on page 1108
- “Connecting to an EIS by using the Adapter Connection wizard” on page 1108
- “Configuring WebSphere Adapters nodes for secondary adapters” on page 1110
- “Calling new services from a WebSphere Adapters request node without changing existing deployed resources” on page 1111

- “Handling new event types from a Websphere Adapters input node without changing existing deployed resources” on page 1112
- “Interacting with an SAP application” on page 1112
- “Interacting with a Siebel application” on page 1122
- “Interacting with a PeopleSoft application” on page 1126
- “Interacting with a JD Edwards application” on page 1129

WebSphere Broker Adapters Transport

WebSphere Broker Adapters Transport is a service that connects applications to Enterprise Information Systems (such as SAP Software, PeopleSoft Enterprise, and Siebel Business Application systems).

You can use the WebSphere Broker Adapters Transport to support the following operations:

- Accept input from an SAP, Siebel, or PeopleSoft application.
- Send requests to an SAP, Siebel, JD Edwards, or PeopleSoft application.
- Send a reply to an SAP synchronous callout.
- Route IDocs to separate message flows.

These operations are implement by the following built-in nodes:

- SAPInput node
- SAPRequest node
- SAPReply node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node
- JDEdwardsRequest node

Use the input nodes to accept input from an EIS, along with a matching reply node if the EIS needs a response. Use the request nodes to send requests to an EIS.

To use the WebSphere Broker Adapters Transport, you must deploy a message flow that contains one or more WebSphere Adapters nodes.

The following topics contain more information about working with WebSphere Adapters:

- “Connecting to Enterprise Information Systems” on page 1049
- “WebSphere Adapters nodes” on page 1050
- “Developing message flows that use WebSphere Adapters” on page 1106

WebSphere Adapters nodes

A WebSphere Adapters node is a message flow node that is used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft.

The following terms are associated with WebSphere Adapters:

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

EMD
: Enterprise Metadata Discovery. A specification that you can use to examine an EIS and get details of business object data structures and APIs. An EMD stores definitions as XML schemas by default,

and builds components that can access the EIS. In IBM App Connect Enterprise you use the Adapter Connection wizard to examine an EIS.

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

The WebSphere Adapters support two modes of communication:

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

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

The WebSphere Adapters nodes also need XML Schema Definitions (XSD) to ensure that the IBM App Connect Enterprise messages that are propagated to and from the nodes reflect the logical structure of the data in the EIS.

SAP, Siebel, JD Edwards, and PeopleSoft adapters are supported by the following message flow nodes in IBM App Connect Enterprise:

- SAPInput node
- SAPRequest node
- SAPReply node
- SiebelInput node
- SiebelRequest node
- PeopleSoftInput node
- PeopleSoftRequest node
- JDEdwardsRequest node
- JDEdwardsInput node

You can configure the WebSphere Adapters nodes by using properties on the nodes, or by using a policy. For example, you can use an SAP Connection policy to specify the connection details for an SAP system. For more information, see Policy properties.

To effectively maintain the pool of connections to the EIS, you can set a connection timeout value on a policy. For more information, see “Configuring EIS connections to expire after a specified time” on page 213.

The SAPRequest node can also use an identity that is present on an input message, and propagate it to SAP, by using the `Propagate` property on the security profile that is defined on the node. For more information, see “Identity and security token propagation” on page 2580.

To maximize performance and avoid unnecessary data conversion, ensure that messages that are passed to a WebSphere Adapters request node contain the correct data types. The DataObject domain is the default domain when parsing messages that are produced by the WebSphere Adapters request node. However, when passing data to the request node (for example, by using an MQInput node), the use of a different domain can improve performance. For example, use the XMLNSC parser with the MQInput node to parse XML messages.

The mode in which your integration node is working can affect the number of integration servers and message flows that you can deploy, and the type of node that you can use. For example, in Remote Adapter Deployment mode, only adapter-related features are enabled, and the types of node that you can use, and the number of integration servers that you can create, are limited. For more information about the available modes of operation, see "Operation modes" on page 3.

For more information about support for adapters on different operating systems, see IBM App Connect Enterprise system requirements.

The following topics provide an overview of the WebSphere Adapters:

- "Overview of WebSphere Adapter for SAP Software" on page 1052
- "Overview of WebSphere Adapter for Siebel Business Applications " on page 1092
- "Overview of WebSphere Adapter for PeopleSoft Enterprise" on page 1098
- "Overview of WebSphere Adapter for JD Edwards EnterpriseOne" on page 1102

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

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

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

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

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

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

For more information, see "Technical overview of Adapter for SAP Software" on page 1052.

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

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

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

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

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

Overview of the outbound processing interfaces

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

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

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

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

    This interface is useful when the event sequence must be preserved.
- The Query interface for SAP Software retrieves data from specific SAP application tables. It can return the data or check for the existence of the data. You can use this type of interaction with SAP if you need to retrieve data from an SAP table without using an RFC function or a BAPI.
- With the Application Link Enabling (ALE) interface, you exchange data using SAP Intermediate Data structures (IDocs). For outbound processing, you send an IDoc or a packet of IDocs to the SAP server.

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

- With the Advanced event processing interface, you send data to the SAP server. The data is then processed by an ABAP handler on the SAP server.

Overview of the inbound processing interfaces

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

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

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

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

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

  If you select assured once-only delivery, the adapter uses a data source to persist the event data, and event recovery is provided to track and recover events in case a problem occurs when the adapter attempts to deliver the event to the endpoint.
- With the ALE pass-through IDoc interface, the SAP server sends the IDoc through the adapter to the endpoint with no conversion of the IDoc. The message tree contains a BLOB field that represents the IDoc.
- The Advanced event processing interface polls the SAP server for events. It discovers events waiting to be processed. It then processes the events and sends them to the endpoint. For more information, see "The Advanced event processing interface" on page 1086.

How the adapter interacts with the SAP server

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

For more information, see the following topics.

- "The Adapter Connection wizard (SAP)" on page 1055
- "The BAPI interfaces" on page 1055
- "The ALE interfaces" on page 1074
- "Query interface for SAP Software" on page 1084
- "The Advanced event processing interface" on page 1086

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

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

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

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

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

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

The BAPI interfaces
The WebSphere Adapter for SAP Software supports outbound processing for simple BAPIs, BAPI units of work, and BAPI result sets. In outbound processing, message flows call BAPIs and other RFC-enabled functions on the SAP server. The adapter supports inbound processing for simple BAPIs only. In inbound processing, the SAP server sends an RFC-enabled function (such as a BAPI function) through the adapter to an endpoint.

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

BAPI interface (simple BAPIs)

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

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

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

- In asynchronous tRFC inbound processing, the adapter does not have to be available when the SAP server runs the function call. The function call is placed on a list of functions to be invoked, and the call is attempted until it is successful.

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

BAPI work unit interface

A BAPI work unit consists of a set of BAPIs that are processed in sequence to complete a task. For example, to update an employee record in the SAP system, the record needs to be locked before being updated. This task is accomplished by calling three BAPIs, in sequence, in the same work unit. The following three BAPIs illustrate the kind of sequence that forms such a unit of work:

- BAPI_ADDRESSEMP_REQUEST
- BAPI_ADDRESSEMP_CHANGE
- BAPI_ADDRESSEMP_APPROVE

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

BAPI result set interface

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

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

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

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

For more information, see the following topics.

- "Outbound processing for the BAPI interface" on page 1056
- "Business objects for the BAPI interface" on page 1073

Note: It is recommended that you modify the mapping from the Adapter to the SAP node when you migrate from a previous version of the Adapter for SAP to the current version of the Adapter for SAP. This precaution prevents empty tags in the BAPI response structure for uninitialized table parameters.

Outbound processing for the BAPI interface
In BAPI outbound processing, a message flow sends a request to the SAP server. For BAPI units of work and BAPI result sets, processing is handled synchronously (the message flow waits for a response

from the SAP server). For simple BAPIs, you can request that processing be handled synchronously or asynchronously (the message flow does not wait for a response from the SAP server).

For BAPI units of work and BAPI result sets, the processing is handled as described in "Synchronous RFC" on page 1057. For simple BAPIs, you make a selection, during configuration, about the type of remote RFC call you want to make.

Synchronous RFC

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

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

Asynchronous transactional RFC

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

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

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

Asynchronous queued RFC

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

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

4. The adapter uses the tRFC protocol to make the call to the specified queue on the SAP server.

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

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

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

Information about the state of in-flight transactions is held in IBM MQ queues, which means that IBM App Connect Enterprise must have access to IBM MQ when it is processing these messages. To enable this, ensure that either an IBM MQ client or an IBM MQ server is installed on the same computer as the integration node. By default, the `Transaction mode` property of the SAPInput node is set to `Yes`, and the `Transaction mode` property of the SAPRequest node is set to `Automatic`, which means that it uses the value that is set on the input node that drives the message flow. If you do not require the nodes in the flow to be transactional, you can set the `Transaction mode` property to `No`, in which case access to IBM MQ is not required. For more information about using IBM MQ with IBM App Connect Enterprise, see “Installing IBM MQ” on page 96.

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

BAPIs with implicit commit

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

Message flow transactionality

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

If the `Transaction mode` property on the SAPRequest node is set to `No`, the adapter does not issue an SAP commit and the parameter that you set on the Adapter Connection wizard has no relevance. However, the commit can still be issued as part of a BAPI work unit COMMIT verb (to which the property on the wizard does apply) or a call to the BAPI_TRANSACTION_COMMIT (to which the property on the wizard does not apply).

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

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

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

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

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

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

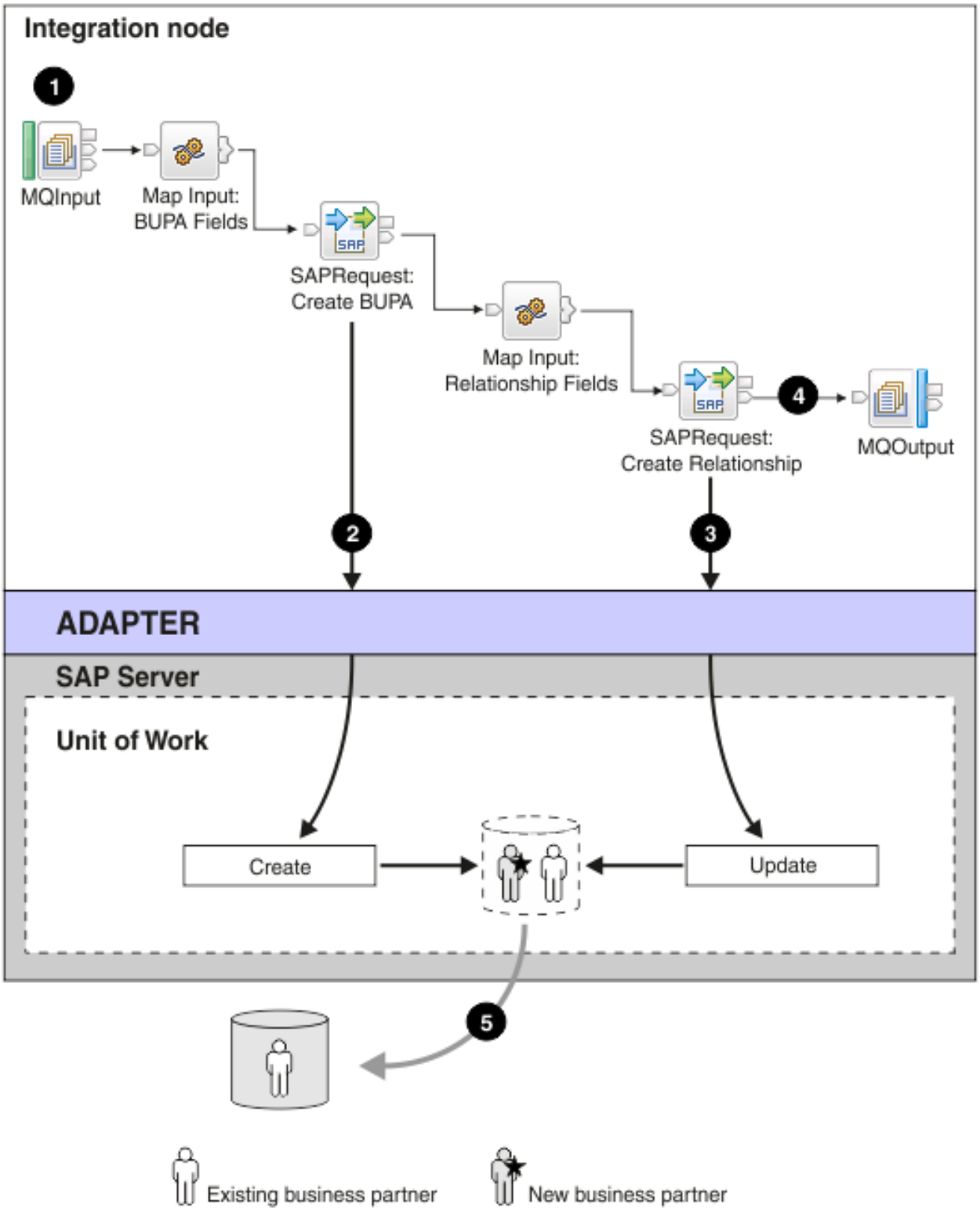
Scenario 1: Business partner and relationship processing in a single flow
You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing in a single message flow.

This scenario is one of two examples that illustrate the concepts that are described in “SAP BAPI transaction commit” on page 1058; see also “Scenario 2: Order create and query application processing with two flows” on page 1061.

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

```
BAPI_BUPA_CREATE_FROM_DATA
BAPI_BUPR_RELATIONSHIP_CREATE
```

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



1. An application triggers the transactional flow that creates the business partner.
2. The SAPRequest node submits a BUPA creation and returns the business partner number. The commit happens when the message flow completes because the node participates in a message flow level transaction.

3. The second SAPRequest node attempts to create a relationship between an existing business partner and the new business partner; however, SAP has not yet committed the creation of the new business partner to the database.

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

   If the `Transaction mode` property were set to `Yes` on the create BUPA call, but `No` on the create relationship call, the adapter would need to use two different connections to SAP; that is, the transactional properties of the connections would be different. The create relationship call would therefore fail because the new business partner would not be visible until the message flow and the transactional commit have completed.
4. The MQOutput node puts an MQ message on the output queue pending transactional commit.
5. The message flow completes and the integration node begins to commit all the resources involved in that flow, including SAP (5 in the diagram). The updates are committed in SAP.

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

Scenario 2: Order create and query application processing with two flows
You must set the `Transaction mode` property appropriately on an SAPRequest node when you are processing by using separate message flows.

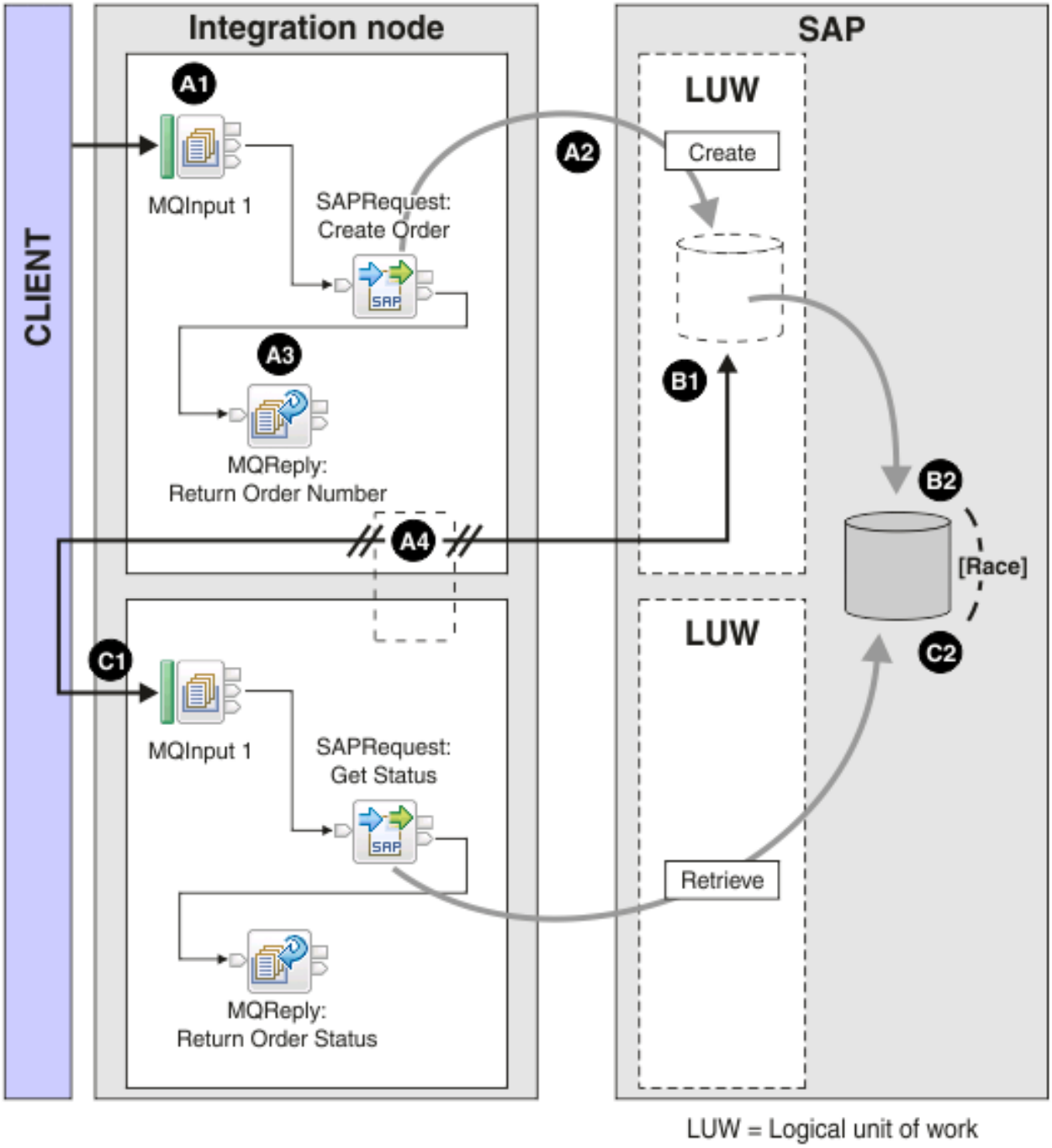
This scenario is one of two examples that illustrate the concepts that are described in “SAP BAPI transaction commit” on page 1058; see also “Scenario 1: Business partner and relationship processing in a single flow” on page 1059.

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

```
BAPI_SALESORDER_CREATEFROMDAT2
BAPI_SALESORDER_GETSTATUS
```

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

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



BAPI Create Order message flow

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

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

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

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

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

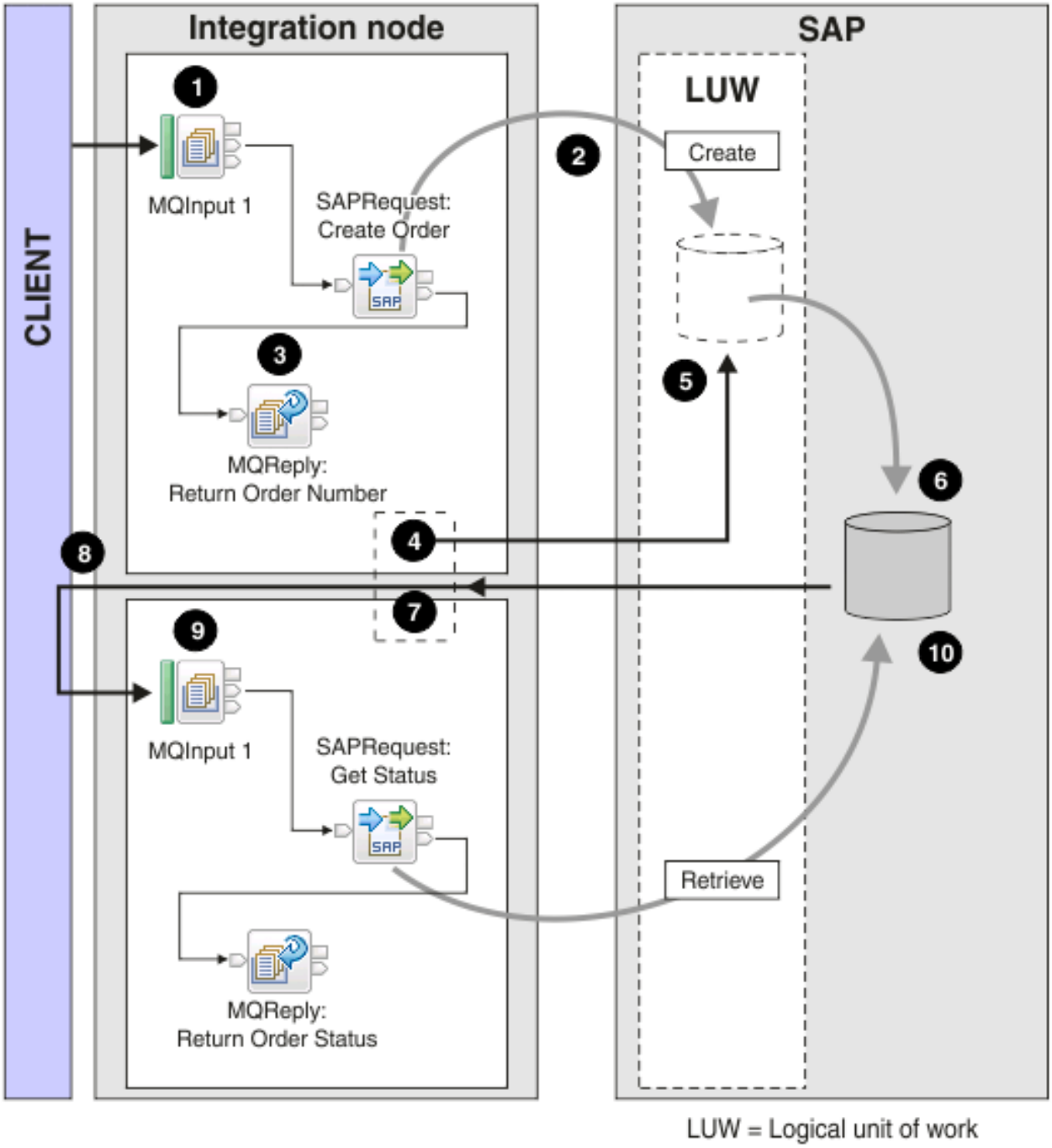
SAP Commits processing asynchronously	BAPI Get Order Status message flow
B1. The SAP commit begins.	**C1.** A request for an order status query is made.
B2. The SAP commit completes.	**C2.** The SAPRequest node requests the order.

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

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

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

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



BAPI Create Order message flow

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

SAP Commits processing synchronously

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

BAPI Get Order Status message flow

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

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

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

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

For more information, see the following topics:

- “Synchronous and asynchronous RFC ” on page 1065
- “Event recovery for the BAPI interface” on page 1066
- “Passing parameters and reporting errors” on page 1066
- “BAPI inbound scenarios” on page 1067
- “SAP adapter scalability and performance” on page 1071

Synchronous and asynchronous RFC
For BAPI inbound and outbound processing, you can specify that the processing be handled synchronously (in which both the message flow and the adapter must be available during processing) or asynchronously (in which the adapter does not have to be available when the message flow makes the function call). In synchronous processing, the message flow waits for a response from the adapter. In asynchronous processing, the SAP application does not wait for a response and the adapter does not have to be available when the SAP application makes the function call.

For diagrams that illustrate synchronous and asynchronous Remote Function Calls (RFC), see “BAPI inbound scenarios” on page 1067.

The BAPI interface has two sets of activation specification properties (one for synchronous RFC and one for asynchronous RFC), which you use to set up inbound processing. You specify values for the properties with the Adapter Connection wizard.

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

Synchronous RFC

If you select **Synchronous RFC** (the default) during configuration, the following processing steps occur:

1. The adapter starts event listeners, which listen for an RFC-enabled function event (which you specified with the RFCProgramID property) on the SAP server.
2. The RFC-enabled function event is pushed to the adapter by way of the event listeners.
3. The adapter resolves the operation and business object name using the received RFC-enabled function name.
4. The adapter sends the business object to an endpoint in a synchronous manner.
5. The adapter receives the response business object from the endpoint.

6. The adapter maps the response business object to an RFC-enabled function and returns it to the SAP server.

The adapter does not listen for events until the endpoint is active and available.

Asynchronous transactional RFC

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

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

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

   A transaction ID is associated with the call.

   The calling program on the SAP server does not wait to see whether the call to the adapter was successful, and no data is returned to the calling program.
2. The RFC-function call is placed on a list of functions to be delivered.

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

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

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

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

   If a failure occurs when the adapter attempts to deliver the business object, the transaction ID remains in the event table. When another event is received from the SAP server, the following processing occurs:

   a. The adapter checks the transaction ID.

   b. If the event matches an ID in the table, the adapter processes the failed event once; it does not send the event with the duplicate ID, thereby ensuring that the event is processed only once.

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

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

Passing parameters and reporting errors
The BAPI interface is defined by its input parameters (IMPORT), output parameters (EXPORT), and tables.

After the Adapter Connection wizard has discovered the BAPI interface, the wizard creates a message set that contains an element and a type for the definition of that interface. Each of the import or export

parameters has a corresponding field, which has an associated type that can be simple or complex. Tables are represented in the message type definition as repeating complex structures (maxOccurs = -1).

BAPIs are also defined by the messages (error, warning, or success) that they can return. These messages are typically returned in an export parameter (for example, BAPIRETURN). Most BAPIs have this parameter in common, although the type of the parameter can vary. For example, its type can be:

- BAPIRETURN
- BAPIRET1
- BAPIRET2

These structures are similar, except for a few fields that have been added in later versions.

To send an error back to the calling SAP program, use the BAPIReturn structure that was specified as one of the export parameters when the BAPI was defined. Messages are returned in the Return export parameter. You can use the transaction code SE91 for message maintenance.

For more information, see Return Parameters (Error Handling) in the BAPI Programming Guide Reference on the SAP Help Portal.

As well as the application level errors, which can be reported by the Return export parameter, system or communication failures also exist, which indicate to SAP that the function could not be called or did not complete.

BAPI inbound scenarios
In SAP, you can call functions in other applications or SAP systems that are registered with SAP as remote function call (RFC) servers. In IBM App Connect Enterprise, you can register the SAP adapter with SAP as an RFC server so that it accepts synchronous and asynchronous calls from SAP.

Integrating SAP with a system with a synchronous interface

In SAP, a BAPI interface is a function call. If a system is connected to SAP as an RFC server, you can use the BAPI interface to define the interface where a program running in SAP can call an external system. The external system is identified in SAP by its RFCDestination value, which is bound to a program ID in SAP administration. The program ID is specified by the external system when it first connects to SAP.

By using the WebSphere Adapter for SAP, a message flow can connect to SAP as an RFC Server by configuring the adapter with the relevant program ID and deploying the message flow to the integration node. After deployment, the message flow can receive synchronous function calls by using the BAPI interface.

The reply identifier

The BAPI import parameters are received by the adapter and propagated from an SAPInput node as a message tree structure. The BAPI export parameters are propagated to an SAPReply node as a message tree structure. The adapter then sends back the export parameters to the calling SAP program. In this case, the SAPReply node is typically in the same flow as the SAPInput node. The SAPInput node provides a unique ID (the reply identifier) for each BAPI call. The reply identifier is propagated to the SAPReply node (in the local environment) to indicate to which BAPI call it is replying.

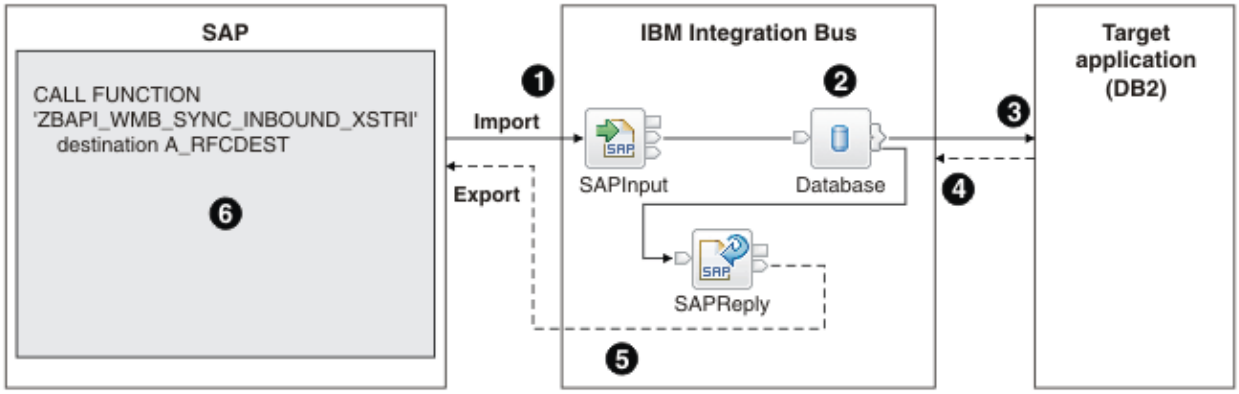
If no reply identifier exists in the local environment, the SAPReply node automatically uses the reply identifier from the SAPInput node that triggered the current execution of the message flow. If the flow is triggered by anything but an SAPInput node, or if a break occurs in the flow, an error is issued. An unhandled exception in the message flow causes a system failure in SAP.

Even if the BAPI does not expect any output parameters, the (empty) message must be propagated to the SAPReply node.

You can process two concurrent callouts from SAP by configuring the message flow with an additional instance.

Scenario 1

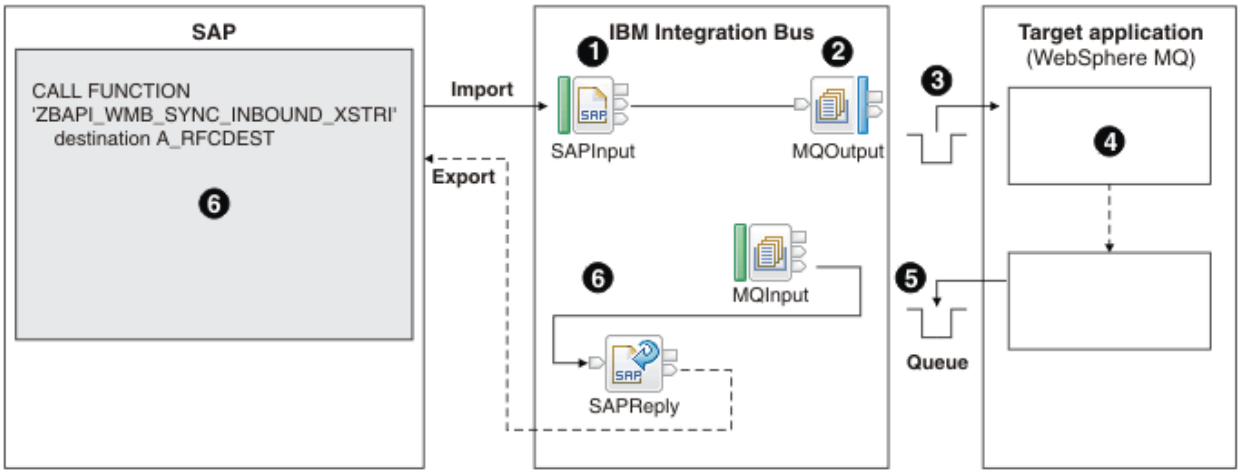
The following diagram represents a message flow where IBM App Connect Enterprise provides a link between SAP and a target application (in this case, DB2) The SAP program requires reply data, therefore it blocks further processing until the call completes.



1. The SAP program makes a BAPI call to IBM App Connect Enterprise.
2. IBM App Connect Enterprise converts the call to an SQL call.
3. IBM App Connect Enterprise passes the call on to DB2.
4. DB2 processes the SQL and returns the result to IBM App Connect Enterprise.
5. IBM App Connect Enterprise converts the SQL result to a BAPI reply and sends the reply to SAP.
6. The SAP program processes the next line of code.

Scenario 2

The following diagram also represents a message flow where the SAP program requires reply data, but in this scenario, calls between IBM App Connect Enterprise and the target application (in this case, IBM MQ) are asynchronous. The SAP system blocks further processing until the call completes. If the value of the SAP **`Maximum client wait time`** property is insufficient, or if it is set to `0`, then a timeout error might be generated. For more information about the **`Maximum client wait time`** property of the SAP Input Node, see Maximum client wait time (secs). When you use the SAPReply node in a different flow from the SAPInput node, deploy the SAPReply node in the same integration server as the SAPInput node.

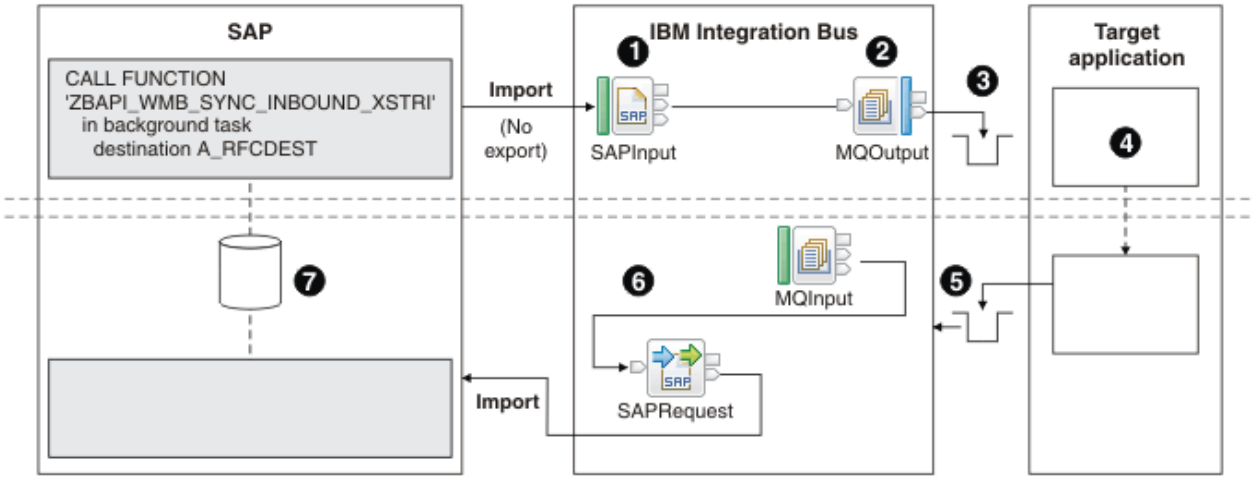


1. The SAP program makes a BAPI call to IBM App Connect Enterprise.
2. IBM App Connect Enterprise converts the import parameters into a message format that is understood by the target application.
3. IBM App Connect Enterprise puts that message on a request queue.

4. The target application gets the request message from the queue, processes it, and puts a reply message on the reply-to queue.
5. IBM App Connect Enterprise gets the reply message from the queue.
6. IBM App Connect Enterprise converts the reply message to BAPI export parameters and sends the reply to SAP.

Scenario 3

The following diagram represents an asynchronous call from SAP to IBM App Connect Enterprise, and an asynchronous call from IBM App Connect Enterprise to a target application. This scenario shows how you can combine the inbound processing that is described in this topic with outbound processing to achieve the same result as in scenarios 1 and 2. For more information about outbound processing, see "Outbound processing for the BAPI interface" on page 1056.



1. The SAP program makes a BAPI call to IBM App Connect Enterprise, stores relevant information in a database table, and continues to process the next line of code.
2. IBM App Connect Enterprise converts the import parameters into a message format that is understood by the target application.
3. IBM App Connect Enterprise puts that message on a request queue.
4. The target application gets the request message from the queue and processes it.
5. IBM App Connect Enterprise gets the reply message from the queue.
6. The SAPRequest node sends the message to the SAP program, requesting an update in SAP.
7. The SAP program refers to the information that is stored in the database table and makes the requested update.

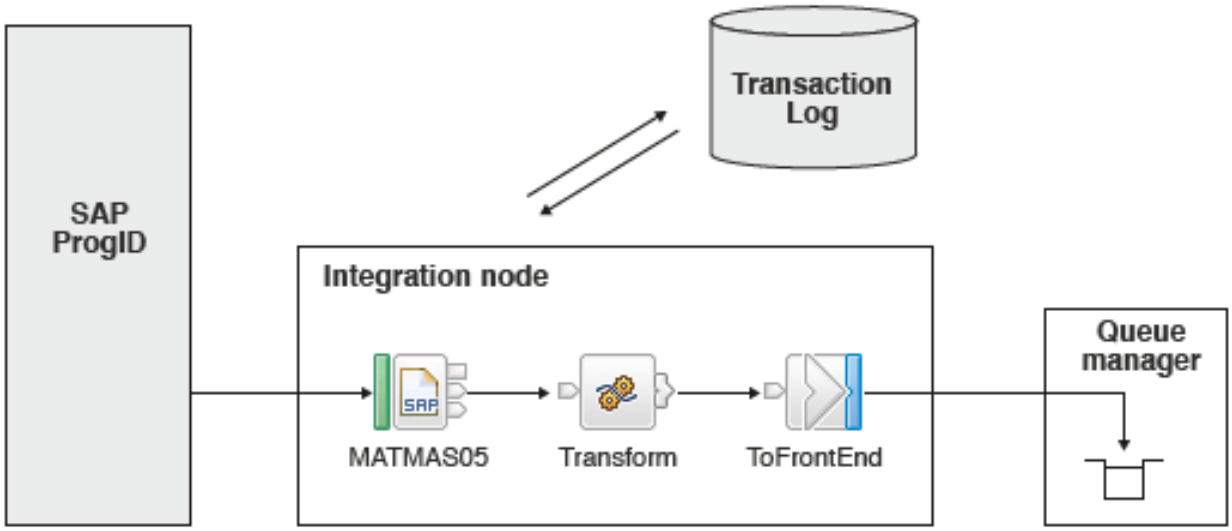
Errors and warnings

- If an SAPReply node is deployed but no SAPInput node is deployed to that integration server, a warning is written to syslog or the Windows Event Viewer.
- If the SAPReply node is provided with a reply identifier that does not correspond to any BAPI call in this integration server, an error is issued.
- If the same reply identifier is sent to two SAPReply nodes, the second node receives an error message.
- If an SAP program tries to call a BAPI that is using the RFCDestination value of the integration node, but that BAPI was not discovered for that adapter, an error is written to syslog or the Windows Event Viewer, and a failure message is sent back to the calling SAP program.

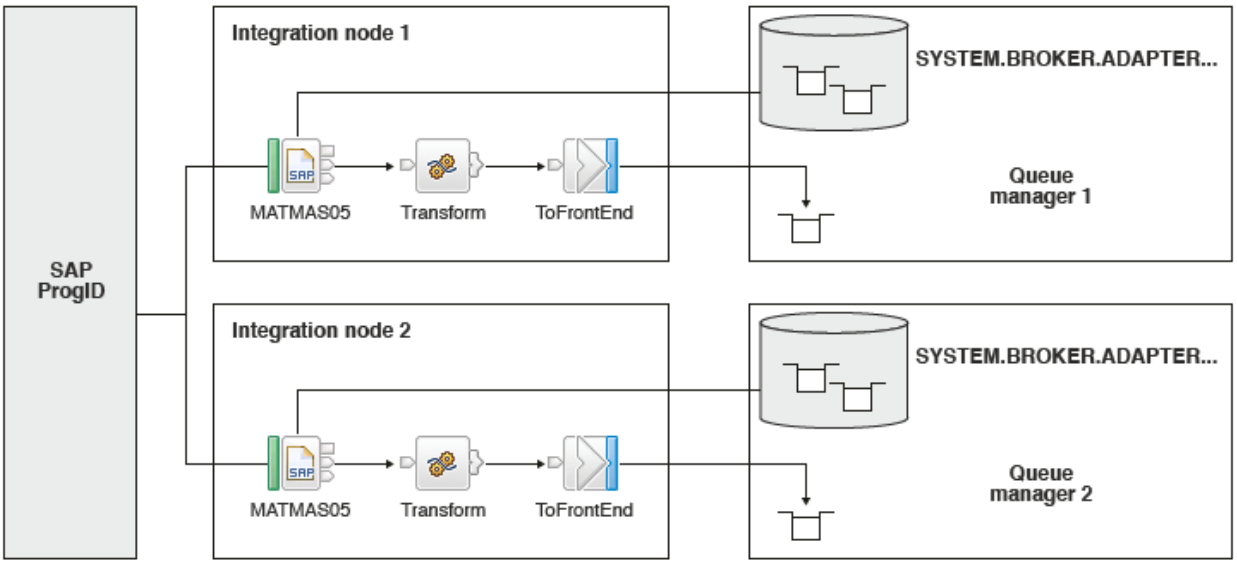
SAP high availability

You can configure IBM App Connect Enterprise to withstand software or hardware failures when working with SAP, so that IBM App Connect Enterprise is available for as much of the time as possible.

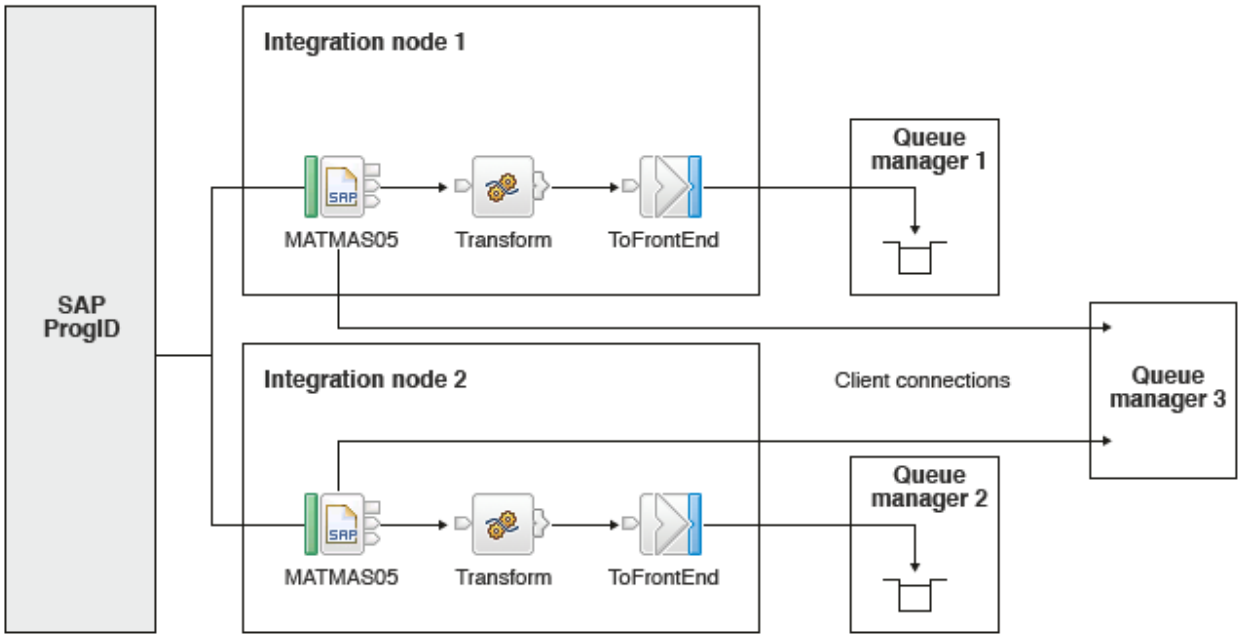
In previous versions of IBM App Connect Enterprise, the tRFC protocol between SAP and IBM App Connect Enterprise (acting as the RFC server) ensures that IDocs and tRFC BAPI calls are delivered exactly once. This behavior is possible because each delivery has an associated transaction ID (TID). IBM App Connect Enterprise monitors the progress of a delivery until SAP confirms a successful delivery. If the connection is lost or IBM App Connect Enterprise fails before that confirmation is issued, SAP attempts to redeliver the message. By keeping a persistent record (in the TID store or transaction log), IBM App Connect Enterprise can ensure integrity and avoid a duplicate delivery.



When two `.inadapter` components, with the same RFC program ID, are deployed to two integration nodes, two connections to the same RFC server are visible to SAP. If the connection is lost to one of the integration nodes, SAP might attempt to redeliver to the other integration node. The integration nodes have separate TID stores, therefore, the second integration node accepts the redelivery, even though the first integration node might have processed some (or all) of the IDocs in the packet.



In IBM App Connect Enterprise, you can now move the TID store to a remote queue manager that can be shared between two integration nodes. To avoid a single point of failure, make this third queue manager an IBM MQ high availability, multi-instance queue manager. For instructions, see “Setting up SAP for high availability” on page 1118.



SAP adapter scalability and performance
You can improve performance by configuring the number of listeners on the adapter and the number of additional instances on the message flow to prevent delays when processing synchronous calls from SAP.

The SAP inbound adapter receives synchronous calls from SAP. The adapter has a property called `Number of listeners`, which controls the maximum number of concurrent calls by configuring the adapter to have a particular number of threads listening to the SAP program ID. The listeners send the input parameters of these calls to the SAPInput node for processing, and the output parameters are sent to the SAPReply node.

When the listener receives a call from SAP, it blocks processing until a message flow instance that contains the SAPInput node is available. When a message flow instance has become available, and has started to process the import parameters, the listener again blocks processing until a message that contains the export parameters is propagated to an SAPReply node.

The amount of time that elapses between the SAPInput node sending the message and the SAPReply node receiving the reply message can be affected by the additional instances property on the message flow. To avoid delays in processing, tune the maximum number of listeners and the number of additional instances for the message flow that contains the SAPInput and SAPReply nodes. You can configure the number of additional instances at message flow level or on the SAPInput node itself.

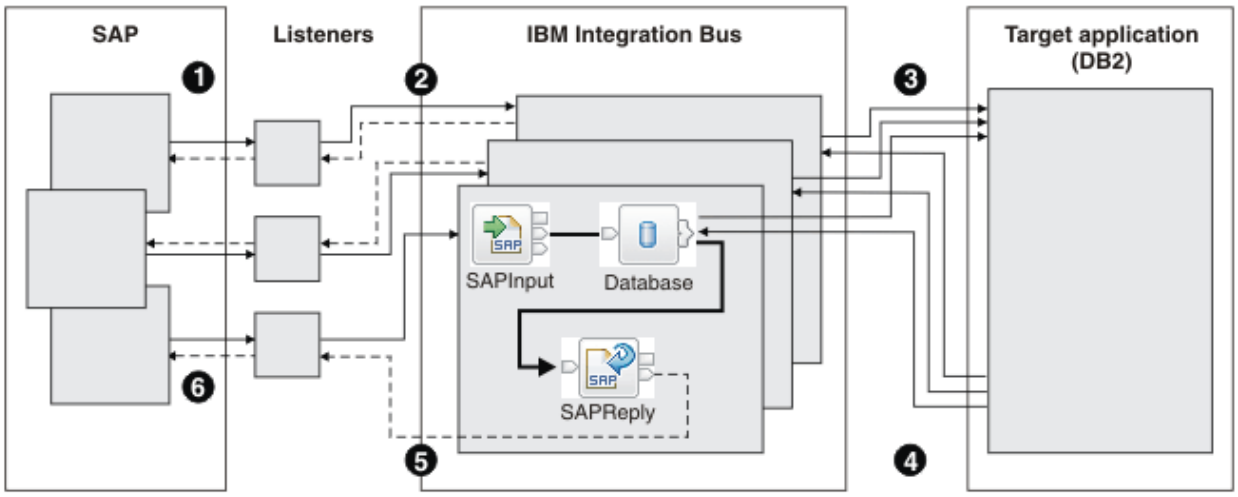
Restrictions

The number of listeners is limited by the SAP configuration. In SAP transaction SMQS, you can view and change the maximum connections property for each RFC destination. Configuring a number of listeners greater than the maximum number of connections has no effect.

For each additional instance of the message flow, extra resources are used by each node in the flow, depending on the types of node in the flow.

Scenarios

The following diagram illustrates a basic scenario, where the number of listeners is equal to the number of additional instances of the message flow; in this case, the scenario has been configured with three listeners and three message flow instances.



1. SAP makes three calls, each of which is received by a listener.
2. Each of the listeners sends the input parameters of each call to the SAPInput node in one of three instances of the message flow.
3. Each message flow instance sends its message to the target application.
4. The target application processes the messages and returns replies to the message flow instances.
5. The SAPReply node in each message flow instance sends the reply message to the listener that received the original call.
6. Each listener returns the reply message to the appropriate SAP program.

The SAPReply node can be in the same message flow as the SAPInput node, as illustrated in the previous example. However, in the following scenario, the SAPReply node is in a different flow from the SAPInput node. The SAPReply node must be deployed in the same integration server as the SAPInput node.

1. SAP makes three calls, each of which is received by a different listener.
2. Each of the listeners sends the input parameters of each call to a message flow that contains an SAPInput node.
3. The message flow puts the message onto a queue for the target application.
4. The target application gets the messages from the queue and processes them.
5. The target application puts the messages onto a queue.
6. A second message flow that contains an SAPReply node gets the messages from the queue and processes them.
7. The SAPReply node sends the reply messages to the appropriate listeners.
8. Each listener returns a reply message to the appropriate SAP program.

In this scenario, the message flow has low latency compared to the time taken by the entire scenario. Therefore, you can limit the resources that are used by the message flow containing the SAPInput node by configuring fewer additional instances for this message flow. One instance of the message flow, as in the example, can service many listeners because the message flow propagates the import parameters quickly for processing.

The following scenarios are also possible.

- A single message flow contains an SAPReply node but it is used to reply to several SAPInput nodes. After the message has been propagated to the SAPReply node, the listener sends the reply back to SAP and is therefore ready to receive another call from SAP. However, the current instance of the message flow is still busy processing the nodes downstream from the SAPReply node. In this case, increase the number of instances on the message flow that contains the SAPReply node.

- After propagating to the SAPReply node, other nodes in the message flow perform extra processing. In this case, increase the number of instances on the message flow that contains the SAPReply node, even when it is the same flow that contains the SAPInput node.

  This scenario might have an undesirable effect on the integrity of your data. If an exception occurs after the SAPReply node, resources that are updated by the message flow (for example, database tables or IBM MQ queues) might be rolled back. However, the reply has been sent back to SAP already and cannot be rolled back. If this behavior is not appropriate, you can improve data integrity by ensuring that the SAPReply node is the final node in the message flow.

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

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

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

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

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

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

How business-object definitions are created

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

Business object structure

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

For more information, see the following topics.

- “Business object structure for a simple BAPI” on page 1073
- “Business object structure for a nested BAPI” on page 1074
- “Business object structure for a BAPI work unit” on page 1074
- “Business object structure for a BAPI result set” on page 1074

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

IDocs

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

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

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

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

- **ALE interface**

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

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

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

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

- **ALE pass-through IDoc interface**

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

Transactional RFC processing

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

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

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

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

Queued RFC processing

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

For more information about ALE interfaces, see the following topics:

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

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

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

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

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

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

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

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

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

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

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

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

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

For more information, see the following topics.

- "Event error handling" on page 1077
- "Event recovery for the ALE interface" on page 1078
- "Event processing for parsed IDoc packets" on page 1078
- "Event processing for unparsed IDocs" on page 1079
- "IDoc status updates" on page 1080

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

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

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

   The adapter uses the activation specification values for RetryLimit and RetryInterval.

   - If the SAP application is not active, the adapter attempts to restart the listeners for the number of times configured in the RetryLimit property.
   - The adapter waits for the time specified in the RetryInterval parameter before attempting to restart the event listeners.
3. If the attempt to restart the event listeners fails, the adapter performs the following actions:

   - The adapter logs the error condition in the syslog (on Linux and UNIX systems), Windows Event Viewer, or user trace log.
   - The adapter cleans up the existing ALE event listeners.
   - The adapter starts new event listeners.

   The adapter uses the values of the RetryLimit and RetryInterval properties when starting the new event listeners.

4. If all the retry attempts fail, the adapter logs the relevant message and CEI events and stops trying to recover the ALE event listener.

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

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

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

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

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

1. The adapter checks the status of the event and takes one of the following actions:
   - If this is a new event, the adapter stores an EVNTID (which corresponds to the transaction ID) along with a status of `0` (Created) in the event recovery table.
   - If the event status is `-1` (Rollback), the adapter updates the status to `0` (Created).
   - If the event status is `1` (Executed), the adapter returns an indication of success to the SAP system.
2. The SAP system sends the IDoc to the adapter.
3. The adapter converts the IDoc to a business object and sends it to the endpoint.

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

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

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

the objects to the appropriate endpoint as part of a unique XA transaction (a two-phase commit transaction) controlled by the application server.

c. When the endpoint receives the event and the transaction is committed, the adapter increments the number in the BQPROC property.

The message endpoint must be configured to support XA transactions.

If the adapter encounters an error while processing a split IDoc packet, it can behave in one of two ways, depending on the IgnoreIDocPacketErrors configuration property:

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

This property is not used for single IDocs and for non-split IDoc packets.

d. The SAP system sends a COMMIT call to the adapter.

e. After the adapter delivers all the business objects in the IDoc packet to the message endpoint, it updates the event status to 1 (Executed).

f. In case of abrupt interruptions during IDoc packet processing, the adapter resumes processing the IDocs from the current sequence number. The adapter continues updating the BQPROC property, even if IgnoreIDocPacketErrors is set to `true`. The adapter continues the processing in case you terminate the adapter manually while the adapter is processing an IDoc packet.

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

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

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

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

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

Unparsed data format

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

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

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

Figure 1. Example of a segment before processing

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

Figure 2. Example of a segment after processing

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

Limitations

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

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

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

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

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

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

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

- SuccessCode
- FailureCode
- SuccessText
- FailureText

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

- Set the UpdateStatus configuration property to `true` and set values for the SuccessCode and FailureCode configuration properties.
- Configure the inbound parameters of the partner profile of the logical system in SAP to receive the ALEAUD message type. Set the following properties to the specified values:

Table 29. Inbound properties of the logical system partner profile	
SAP property	**Value**
Basic Type	ALEAUD01
Logical Message Type	ALEAUD
Function module	IDOC_INPUT_ALEAUD
Process Code	AUD1

Pass-through support for IDocs, and MQSeries link for R/3 link migration
Both the inbound and outbound SAP adapters support a pass-through mode for IDocs.

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

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

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

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

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

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

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

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

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

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

ALE pass-through IDoc business object structure
During ALE processing, the adapter exchanges business objects with the SAP application.

The message tree contains a BLOB field that represents the IDoc. The BLOB field is modelled as a HexBinary type in the XSD file. The message tree contains the stream data (the BLOB field), which is the entire data of the IDoc. The transaction ID, iDoc type, and queue name fields contain information about that data and how it has been transferred.

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

To parse the IDoc control record, select **Parse the IDoc Control Record** when you run the Adapter Connection wizard.

Generic IDoc routing
By using the SAPInput node in passthrough mode, IBM App Connect Enterprise can receive any IDoc and route it according to IDoc type.

The message definition for a generic IDoc in ALE passthrough mode contains four fields. Three of these fields are control information:

- The transaction ID
- The queue name (if qRFC is used)
- The name of the IDoc type

The fourth field is a hexBinary field that contains the entire IDoc in a raw, unparsed format. You can use the DataObject parser to parse this raw format to a full logical structure of a specific IDoc if you provide a library that contains the definition for that IDoc type. To parse raw IDocs, you must use an MQInput node with the `Message domain` property set to `DataObject`, and the `Message format` property set to `SAP ALE IDoc`.

When passthrough mode is not used, the library contains a fully parsed IDoc structure. These structures are specific to each IDoc, therefore the library must have a type defined for every IDoc that could be received by the `.inadapter` component. This library is determined by the RFC Program ID that is configured on the adapter and the ALE or RFC configuration on the SAP side. This behavior can affect how message models are managed.

Whenever you need to develop message flows in isolation, where each flow is to handle a different type of IDoc, the ALE parsed mode is not appropriate because all flows have a common denominator (the library), which needs to be changed whenever a new IDoc type is added.

By using the generic passthrough mode, you can create a routing message flow that uses the IDoc type field of the generic IDoc model to separate IBM MQ queues, for example. You can create message flows that deal with each different IDoc type. If the set of discovered IDocs is extended, you can create a message flow and a library (which contains just the new IDoc), then deploy them, without the need to change existing message flows or libraries.

By using this method, you can also use a single RFC program ID to receive all IDoc types, while still allowing individual IDoc processing.

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

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

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

How business-object definitions are created

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

For more information, see the following topics.

- "ALE business object structure" on page 1083
- "Transaction ID support" on page 1084

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

Wrapper business object

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

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

Note that the transaction ID and queue name attributes are present in the business object even if you are not using the tRFC or qRFC features.

You can generate business object names to match the SAP XI standard by selecting `Generate business objects according to SAP naming conventions`. For more information, see SAP connection properties for the Adapter Connection wizard.

IDoc business object

The IDoc business object contains the following objects:

Control record
The control record business object contains the metadata required by the adapter to process the business object.

The control record can be generated from SAP field names or from SAP field descriptions. While configuring the properties for the control record you can specify if you want the control record to be generated from SAP field names or from SAP field descriptions. If you want the control record generated from field names, select the check box to use SAP field names to generate attribute names.

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

The data record business object is generated for a parsed IDoc. The data record business object contains all the segments of the IDoc. Each segment in turn has a child business object. The segment attributes can also be generated by using SAP field names or field descriptions. You can use SAP field names to generate attribute names.

Dummy keys are not used in IBM App Connect Enterprise.

Unparsed IDocs

For an unparsed IDoc, in which the data part of the IDoc is not parsed by the adapter, the IDoc business object contains a dummy key, a control record, and the IDoc data. The IDoc data is of hexBinary type and represents the whole data record containing segments in binary content.

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

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

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

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

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

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

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

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

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

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

Query interface for SAP software (QISS) supports hierarchical data retrieval from the SAP application tables. By using this interface, the adapter can determine the existence of data in SAP application tables, or retrieve all the data in SAP application tables. For example, to check if customer Bob exists in SAP system, the Adapter Connection wizard can be run to discover the SAP application table KNA1. The wizard then generates the business object for KNA1 along with other SCA service artifacts. At run time, the SAPRequest node passes the KNA1 business object to the adapter to call the QISS interface, the adapter retrieves the table data from SAP, and returns the result to the node.

For more information, see the following topics.

- "Outbound processing for the query interface for SAP Software" on page 1085
- "Business objects for the query interface for SAP Software" on page 1085

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

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

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

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

   The query business object can be in a container business object, or it can be received as a table business object.
2. The adapter determines, from the table object sent with the query, the name of the table to examine.
3. The adapter determines the columns to retrieve or examine.
4. The adapter determines the rows to retrieve or examine.
5. The adapter responds.
   - For a RetreiveAll operation, the adapter returns a result set in the form of a container of query business objects, which represent the data for each row retrieved from the table. If the query is received as a table business object (not inside a container), the rows are returned one at a time, as they are retrieved.
   - For the Exists operation, the adapter returns an indication of whether the data exists in the SAP table.
   - If no data exists, the adapter generates an exception.

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

How data is represented in business objects

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

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

How business objects are created

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

Business object structure

The table business object can be part of a container.

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

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

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

- The sapWhereClause property retrieves information from SAP tables. The default value is populated by the Adapter Connection wizard. The space character is used as the delimiter to parse the sapWhereClause.

- The sapMaxRows property is the maximum number of rows to be returned. The default value is 100.
- The sapRowsSkip property is the number of rows to skip before retrieving data. The default value is 0.

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

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

In the KNA1 business object, notice the reference to SapAdrc, a child business object. The SapAdrc table object has a column named AddressNumber. This column has an associated property (ForeignKey) that contains a reference to the parent business object.

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

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

For inbound processing, it polls for events in SAP, converts them into business objects, and sends the event data as business objects to IBM App Connect Enterprise.

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

For more information, see the following topics.

- "Outbound processing for the AEP interface" on page 1086
- "Inbound processing for the AEP interface" on page 1089
- "Business objects for the Advanced Event Processing (AEP) interface" on page 1092

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

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

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

For more information, see the following topics.

- "ABAP handler overview" on page 1087
- "ABAP handler creation" on page 1087
- "Call Transaction Recorder wizard" on page 1088

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

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

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

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

- Call Transaction

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

- Batch data communication (BDC)

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

- ABAP SQL

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

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

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

- ABAP Function Modules and Subroutines

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

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

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

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

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

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

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

The following table provides information about the parameters:

Table 30. Interface parameters

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

Call Transaction Recorder wizard

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   The property Poll Quantity determines the maximum number of events returned for a single poll call.
5. For each event, the adapter uses the remote function specified for the Retrieve operation to retrieve the data and send it to the endpoint.

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

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

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

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

For more information, see the following topics.

- "Event detection" on page 1090
- "Event triggers" on page 1091

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

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

Adapter-supported event detection mechanisms

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

- Custom Triggers, which are implemented for a business process (normally a single SAP transaction) by inserting event detection code at an appropriate point in the SAP transaction
- Batch programs, which involve developing an ABAP program containing the criteria for detecting an event
- Business workflows, which use the object-oriented event detection capabilities of SAP
- Change pointers, a variation of business workflows, which use the concept of change documents to detect changes for a business process

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

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

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

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

Event table

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

Table 31. Event table fields

Name	Type	Description
event_id	NUMBER	Unique event ID that is a. primary key for the table
object_name	STRING	Business object name.
object_key	STRING	Delimited string that. contains the keys for the business object

Table 31. Event table fields (continued)

Name	Type	Description
object_function	STRING	Operation corresponding to. the event (Delete, Create, or Update)
event_priority	NUMBER	Any positive integer to. denote the priority of the event
event_time	DATE	Date and time when the. event was generated
event_status	NUMBER	
Xid	STRING	Unique XID (transaction. ID) value for assured-once delivery
event_user	STRING	User who created the event.
event_comment	STRING	Description of the event.

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

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

- /CWLD/ADD_TO_QUEUE

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

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

Both functions are for real-time triggering.

Current event table

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

Future event table

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

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

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

Business objects for the Advanced Event Processing (AEP) interface
During advanced event processing, the adapter exchanges business objects with the SAP application.

How data is represented in business objects

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

How business-object definitions are created

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

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

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

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

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

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

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

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

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

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

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

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

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

For more information, see the following topics.

- "Outbound processing for Siebel Business Applications" on page 1093
- "Inbound processing for Siebel Business Applications" on page 1094
- "Business objects for Siebel Business Applications" on page 1097
- "Adapter Connection wizard (Siebel)" on page 1097

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

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

1. The adapter extracts metadata from the WebSphere business object hierarchy.
2. It identifies the appropriate Siebel objects to access (for example, Siebel business objects and business components, or Siebel business services, integrations objects, and integration components) depending on the objects against which the artifacts were generated.
3. The adapter extracts the outbound operation to perform from the WebSphere business object hierarchy.
4. After accessing the required Siebel objects, the adapter retrieves, updates, deletes, or creates a Siebel business component hierarchy or performs the corresponding business service method on the integration component hierarchy.
5. If there are updates (Create, Update, Delete), the adapter populates that Siebel object (business or integration component hierarchy) with data from the hierarchy of WebSphere business objects.

Supported Outbound Operations

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

Table 32. Supported outbound operations

Operation	Description
Create	Creates the business component.
Delete	Deletes the business component and its children.
Exists	Checks for the existence of incoming business objects. The output business object, "ExistsResult" will be returned with the boolean value populated.
Retrieve	Specifies the value of the business component.

Table 32. Supported outbound operations (continued)	
Operation	**Description**
RetrieveAll	Retrieves multiple instances of the same business component and populates it as the container object.
Update	Updates the Siebel application with the incoming business object.

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

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

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

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

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

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

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

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

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

During inbound processing, the adapter polls the event business components from the event store at regular intervals. Each time it polls, a number of events are processed by the adapter. Events are processed in ascending order of priority and ascending order of the event time stamp. In each poll cycle, new events are picked up. The adapter retrieves the value set in the object key field for the event

and loads the business object that corresponds to it. The business object is created from the retrieved information and is delivered to the export components.

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

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

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

Event store structure for Siebel business objects and business components

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

Table 33. Event store structure for `IBM2` Siebel event business objects and business components

Field	Description	Example
Description	Any comment associated with the event.	Account Create Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: `1-XYZ`)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name-value pair consisting of the name of the property (key name) and value.	`ID=1-20RT`
Object name	The name of the business object for which the event was detected.	`IOAccountPRMANIICAccount`
Priority	The event priority.	1

Table 33. Event store structure for IBM2 Siebel event business objects and business components (continued)		
Field	**Description**	**Example**
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0
XID	The transaction ID. This is to ensure assured once-only delivery.	None
CONNECTOR ID	The ID set for the adapter.	Siebel001

Event store structure for Siebel business services

The event is retrieved from the IBM2 event business component and the information is used to retrieve the event business component.

Table 34. Event store structure for IBM2 Siebel business services		
Field	**Description**	**Example**
Description	Any comment associated with the event.	Account PRM ANI Event
Event ID	The ID of the event row.	Automatically generated unique ID in Siebel (for example: 1-XYZ)
Event timestamp	The time stamp for the event. The format is in *mm/dd/yyyy hh:mm:ss*	*02/24/2007 11:37:56*
Event type	The type of event.	Create, Update, or Delete
Object key	A unique identifier of the business-object row for which the event was created. It is a name value pair consisting of the name of the property (key name) and value.	Name=TestName;Location=BGM, where 'Name' and 'Location' are the keys in the integration component. 'TestName' and 'BGM' are the values specified, and ; is the event key delimiter.
Object name	The name of the business object for which the event was detected.	IOAccountPRMANIICAccount
Priority	The event priority.	1

Table 34. Event store structure for IBM2 Siebel business services (continued)		
Field	**Description**	**Example**
Status	The event status. This is initially set to the value for a new event and updated by the adapter as it processes the event. The status can have one of the following values: • 0: Identifies a new event. • 1: Identifies an event that has been delivered to an export component. • -1: An error occurred while processing the event. This column cannot contain a null value.	0
XID	The transaction ID. This is to ensure 'assured once delivery'.	None

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

How business objects are created

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

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

Business object structure

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

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

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

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

Overview of WebSphere Adapter for PeopleSoft Enterprise
With the adapter for PeopleSoft Enterprise you can create integrated processes that exchange information with PeopleSoft Enterprise through a standard interface. This interface shields the message flow developer from having to understand the lower level details about implementation of the application or data structures used on the PeopleSoft Enterprise server.

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

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

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

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

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

For more information, see "Technical overview of the WebSphere Adapter for PeopleSoft Enterprise" on page 1098.

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

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

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

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

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

For more information, see the following topics.

- "Outbound processing for PeopleSoft Enterprise" on page 1099
- "Inbound processing for PeopleSoft Enterprise" on page 1099
- "Business objects for PeopleSoft Enterprise" on page 1101

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

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

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

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

Supported outbound operations

WebSphere Adapter for PeopleSoft Enterprise supports the following outbound operations:

Table 35. Supported outbound operations

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

Inbound processing for PeopleSoft Enterprise
The WebSphere Adapter for PeopleSoft Enterprise supports inbound event processing.

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

To use inbound event processing, you must create a custom event project in PeopleSoft, as described in "Creating a custom event project in PeopleTools" on page 1127.

For more information, see "Event store for PeopleSoft Enterprise" on page 1100.

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

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

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

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

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

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

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

Table 36. Event status values

Status short name	Description	Event table value
Error processing event	An error. occurred during event processing	-1
Ready for poll	The event has. The event is not yet beenready to be picked up by thepicked up. adapter	0
Success	The event has. been delivered to the event manager	1

Table 36. Event status values (continued)		
Status short name	**Description**	**Event table value**
Deleted	The event has. been processed successfully and is removed from the event store	4
Future Events	These events. should be processed at a future date	99

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

How business objects are created

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

Business object structure

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

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

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

Attribute property	Description
Cardinality	Single cardinality for simple attributes; multiple cardinality for container attributes.

Overview of WebSphere Adapter for JD Edwards EnterpriseOne
With the WebSphere Adapter for JD Edwards EnterpriseOne, you can create integrated processes that include the exchange of information with a JD Edwards EnterpriseOne server, without special coding.

The adapter provides a standard interface that eliminates the need for the component to understand the lower-level implementation details or data structures of the application. By using the adapter, a component (the program or piece of code that performs a specific business function) can send requests to the JD Edwards EnterpriseOne server (for example, to query a customer record in a table or to update an order document).

WebSphere Adapter for JD Edwards EnterpriseOne complies with the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA). JCA standardizes the way in which application components, application servers, and Enterprise Information Systems (EIS), such as a JD Edwards EnterpriseOne server, interact with each other. WebSphere Adapter for JD Edwards EnterpriseOne makes it possible for JCA-compliant application servers to connect to and interact with the JD Edwards EnterpriseOne server. Clients running on the JCA-compliant server can then communicate with the JD Edwards EnterpriseOne server in a standard way (by using business objects or JavaBeans).

Suppose a medium-sized retail company uses JD Edwards EnterpriseOne to coordinate most of its business operations. JD Edwards EnterpriseOne includes a business function that can return a real-time list of inventory items for its 100 stores located across the United States. An application component might be able to use this business function as part of an overall business process. For example, an employee of a retail company can access the real-time list of available inventory items, therefore providing correct, real-time information to a customer.

For more information, see "Technical overview of the WebSphere Adapter for JD Edwards EnterpriseOne" on page 1102.

Technical overview of the WebSphere Adapter for JD Edwards EnterpriseOne
The IBM WebSphere Adapter for JD Edwards EnterpriseOne provides a way for applications to interact with data on JD Edwards EnterpriseOne applications.

The adapter processes requests using one of two types of business objects: business functions and XML Lists. A business function is a business object container that can contain one or many business objects that can be processed as a single transaction. An XML List is a single business object that can query a table and return multiple records.

You create business objects by using the Adapter Connection wizard. The business objects that are generated by the Adapter Connection wizard have predefined business object definitions.

For more information, see the following topics:

- "Outbound processing (JD Edwards)" on page 1102
- "Inbound processing for JD Edwards EnterpriseOne" on page 1103
- "Business objects for JD Edwards EnterpriseOne" on page 1105
- "The Adapter Connection wizard (JD Edwards)" on page 1105

Outbound processing (JD Edwards)
The WebSphere Adapter for JD Edwards EnterpriseOne supports synchronous outbound request processing. When the adapter receives a request from the module, in the form of a business object, the adapter processes the request and returns the result, when applicable, in a business object.

The adapter processes requests by using one of two types of business object: business functions and XML Lists. A business function is a business object container that can contain one or many business objects, which can be processed as a single transaction. An XML List is a single business object that can query a table and return multiple records. The component sends a request in the form of a WebSphere business

object hierarchy to the adapter, the adapter processes the request, and returns a WebSphere business object hierarchy that represents the result of the operation.

When the adapter receives a WebSphere business object hierarchy, the adapter processes it in the following way.

1. The adapter extracts metadata from the WebSphere business object hierarchy.
2. The adapter identifies the appropriate JD Edwards EnterpriseOne objects to access (for example, JD Edwards EnterpriseOne business objects and business components, or JD Edwards EnterpriseOne business function, or JD Edwards EnterpriseOne XML List) depending on the objects against which the artifacts were generated.
3. The adapter extracts the outbound operation to complete from the WebSphere business object hierarchy.
4. After accessing the required JD Edwards EnterpriseOne objects, the adapter retrieves the data for XML List, or completes the corresponding business function method.
5. If updates are involved (Create, Update, Delete), the adapter populates that JD Edwards EnterpriseOne object (business function or XML List hierarchy) with data from the hierarchy of WebSphere business objects.

Supported outbound operations

When the adapter receives a request, it processes the request by using the JD Edwards EnterpriseOne Dynamic Java connector to call either a business function or an XML List.

Business functions support the following types of operation:

- Create
- Delete
- Execute
- Retrieve
- Update

XML Lists support the retrieveAll operation

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

Inbound processing for JD Edwards EnterpriseOne
The WebSphere Adapter for JD Edwards EnterpriseOne supports asynchronous inbound processing. The adapter polls the JD Edwards EnterpriseOne server at specified intervals for events. When the adapter receives an event, it converts the event data into a business object and sends the business object to the component.

The WebSphere Adapter for JD Edwards EnterpriseOne supports real-time events. A real-time event is a business transaction that provides information from the JD Edwards EnterpriseOne server that can be

used to interoperate with a third-party system. Real-time events can be generated wherever business functions run, such as HTML, WIN32, and enterprise servers. Real-time events are useful for producing notifications in real time. The adapter supports both single and container real-time events.

When the adapter gets a real-time event from the JD Edwards EnterpriseOne transaction server by calling the JD Edwards EnterpriseOne Dynamic Java Connector API, it parses the content of this real-time event and converts it into a business object. The adapter then sends the business object to the event endpoints. For example, if a company is updated, the JD Edwards EnterpriseOne server captures this change immediately, and one real-time event is generated by the JD Edwards EnterpriseOne transaction server. The adapter then communicates with the JD Edwards EnterpriseOne transaction server, retrieves this real-time event, and processes it. After converting it into a business object, the adapter delivers this business object to the event endpoint.

WebSphere Adapter for JD Edwards EnterpriseOne processes events in the following way.

1. The adapter calls the JD Edwards EnterpriseOne Dynamic Java Connector API to get a real-time event.
2. The adapter parses the content of this real-time event.
3. The adapter populates the associated business object with the values that it retrieves from the payload of this real-time event.
4. The adapter sends the generated business object to each registered application.

Before inbound processing can occur, the JD Edwards EnterpriseOne server must be configured to support real-time events.

Event persistence (JD Edwards)
The WebSphere Adapter for JD Edwards EnterpriseOne supports event persistence for inbound processing in case of abrupt termination. Event persistence (or assured-once delivery) is a way to make sure that events are delivered once, and only once, to the endpoint in the case of a failure.

During event processing, the adapter persists the event state in an event store located on the data source. You must set up this data source before you can create the event store. To use the recovery feature, set the `AssuredOnceDelivery` property in the activation specification to `True`. This recovery feature is set to `True` by default.

When a failed event occurs, the adapter tries to deliver the event once more only. If the event fails again, its status is marked as FAILED and the event remains in the event store until it is either deleted manually or has its status changed. If the adapter fails or stops before the event is marked as FAILED, when it recovers, the adapter retrieves the event from the event store with the stored data and attempts to redeliver the event, marking its status as FAILED only after the default retry limit of one has been exceeded again.

Event store (JD Edwards)
The event store is a persistent cache where event records are saved until the polling adapter can process them. The adapter uses event stores to record inbound events as they make their way through the system.

Each time a real-time event is received, the adapter updates the status of the event in an event store. The status of each event is continually updated by the adapter for recovery purposes until the events are delivered to the endpoint. If the adapter detects that no event store exists for the inbound module in IBM App Connect Enterprise, it creates one automatically when the application is deployed to the run time. Each event store that is created by the adapter is associated with a specific inbound module. The adapter does not support multiple adapter modules pointing to the same event store.

When the adapter polls the JD Edwards EnterpriseOne transaction server and receives a real-time event, it creates an entry in the event store for each event that matches the search criteria specified in the activation specification properties. The adapter records the status of each new entry as NEW.

If a real-time event is successfully delivered, the corresponding event store entries are deleted. For failed events, the entries remain in the event store.

Assured once delivery

The JD Edwards EnterpriseOne transaction server provides guaranteed event delivery quality of service. All the real-time events to which the adapter subscribes are delivered to the adapter without any loss. The JD Edwards EnterpriseOne transaction server can send duplicate real-time events to the adapter, therefore the adapter provides assured once event delivery; each event is delivered once and only once. To enable assured once delivery, you must set the `AssuredOnceDelivery` activation specification property to `True`.

When you set the `AssuredOnceDelivery` activation specification property to `True`, the `AutoAcknowledge` activation specification property is set to `False` automatically.

When a real-time event is obtained, it is processed in the following way.

1. The event is delivered to its corresponding endpoint.
2. The event entry is deleted from the event store.
3. An acknowledgment is issued to the JD Edwards EnterpriseOne transaction server.

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

How the adapter uses business objects

The WebSphere Adapter for JD Edwards EnterpriseOne uses the JD Edwards EnterpriseOne Dynamic Java Connector APIs to communicate with the JD Edwards EnterpriseOne server. The adapter exchanges information with JD Edwards EnterpriseOne through business functions, XML List calls, and the real-time event mechanism.

How business objects are created

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

Business object structure

The adapter supports processing of hierarchical business objects. A container business object representing a JD Edwards EnterpriseOne operation is a wrapper object that contains single or multiple child business function objects, also called simple business function objects. Each business function object represents a specific function call in the JD Edwards EnterpriseOne application.

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

See the WebSphere Adapter for JD Edwards EnterpriseOne product documentation online for further information.

By using IBM App Connect Enterprise, you establish a connection to the JD Edwards EnterpriseOne server to browse the metadata repository on the JD Edwards EnterpriseOne server.

You specify connection information (such as the user name and password needed to access the server), and you specify the method of operation that you want to use for an outbound or inbound request:

- For an outbound request, you specify either a business function or an XML List.
- For an inbound request, you specify real-time events.

You can then provide search criteria and select the information (for example, by using the search filter with libraries for business functions, or by selecting tables for XML Lists).

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

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

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

Developing message flows that use WebSphere Adapters

For information about how to develop message flows that use WebSphere Adapters, see the following topics.

About this task

- “Preparing your system to use WebSphere Adapters nodes” on page 1106
- “Activating IBM License Metric Tool for WebSphere Adapters” on page 1108
- “Connecting to an EIS by using the Adapter Connection wizard” on page 1108
- “Configuring WebSphere Adapters nodes for secondary adapters” on page 1110
- “Calling new services from a WebSphere Adapters request node without changing existing deployed resources” on page 1111
- “Handling new event types from a Websphere Adapters input node without changing existing deployed resources” on page 1112
- “Interacting with an SAP application” on page 1112
- “Interacting with a Siebel application” on page 1122
- “Interacting with a PeopleSoft application” on page 1126
- “Interacting with a JD Edwards application” on page 1129

Preparing your system to use WebSphere Adapters nodes

Before you can connect to an Enterprise Information System (EIS), you must prepare your system by adding external software dependencies and configuring the EIS to work with the WebSphere Adapter.

Before you begin

- For general background information, read “WebSphere Adapters nodes” on page 1050
- Check for the latest information about WebSphere Adapters at WebSphere Adapters technotes.
- Check for the latest information about support for adapters on different operating systems at IBM App Connect Enterprise system requirements.
- Enable the WebSphere Adapters nodes in the IBM App Connect Enterprise runtime environment; see “Preparing the environment for WebSphere Adapters nodes” on page 209.
- If you want to use the IBM Tivoli License Manager (ITLM), perform the steps in “Activating IBM License Metric Tool for WebSphere Adapters” on page 1108.

About this task

Perform the following steps, in the order shown, to prepare your system to use WebSphere Adapter nodes.

Procedure

- **SAP**
  1. Follow the instructions in "Adding external software dependencies for SAP" on page 1113.
  2. Follow the instructions in "Configuring the SAP server to work with the adapter" on page 1115.
- **Siebel**
  1. Follow the instructions in "Creating the event store manually" on page 1123.
- **PeopleSoft**
  1. Follow the instructions in "Creating a custom event project in PeopleTools" on page 1127.

What to do next

After you have prepared your system, connect to an EIS by following the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.

Preparing the environment for WebSphere Adapters nodes

Before you can use the WebSphere Adapters nodes, you must set up the IBM App Connect Enterprise runtime environment so that you can access the Enterprise Information System (EIS).

Before you begin

Read "WebSphere Adapters nodes" on page 1050.

About this task

To enable the WebSphere Adapters nodes in the IBM App Connect Enterprise runtime environment, configure the integration server with the location of the EIS provider JAR files and native libraries. (On Windows, the location of the JAR files cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.)

The integration server configuration file (`server.conf.yaml`) contains a section for connector providers, which defines the location of the required JAR files and native libraries, as you can see in the following example:

```
ConnectorProviders:
  SAPConnectorProvider:
    jarsURL: default_Path     # Set to the absolute path containing the SAP JCo JARs
    nativeLibs: default_Path  # Set to the absolute path containing the SAP JCo libraries
```

By default, the location is set to `default_Path`, which indicates that the JAR files and libraries can be found in the integration server's shared-classes directory.

Procedure

To configure the integration server with the location of required JAR files and libraries, complete the following steps.

1. The WebSphere Adapters nodes require libraries from the EIS vendors. For more information on how to obtain and use these libraries, see "Developing message flows that use WebSphere Adapters" on page 1106.
2. Use a YAML editor to open the `server.conf.yaml` file, then find the `ConnectorProviders` section.

   For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
3. Set the `jarsURL` property to the absolute path where the JAR files are stored.
4. Set the `nativeLibs` property to the absolute path where the native libraries are stored.
5. Save the `server.conf.yaml` file.

What to do next

When you have set up the environment for the WebSphere Adapters nodes, you can perform the preparatory tasks that are listed in "Developing message flows that use WebSphere Adapters" on page 1106.

Activating IBM License Metric Tool for WebSphere Adapters

If you want to use IBM License Metric Tool (ILMT), you must activate it for each of the WebSphere Adapters.

About this task

ITLM enables you to monitor the usage of IBM (and other) software products. For more information, see the IBM License Metric Tool website.

The following steps describe how to activate the ILMT file for each of the adapters.

Procedure

1. Locate the ILMT directory for the adapter.
   - For SAP: *install_dir*/itlm/SAP
   - For Siebel: *install_dir*/itlm/Siebel
   - For PeopleSoft: *install_dir*/itlm/PeopleSoft
   - For JD Edwards: *install_dir*/itlm/JDE
2. Remove the inactive file extension from the file in the ILMT directory so that it ends with `.sys2`.

Results

After you have performed these steps, when you run ILMT, the adapter is visible.

Connecting to an EIS by using the Adapter Connection wizard

Use the Adapter Connection wizard to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System (EIS).

Before you begin

- Complete the preparatory tasks listed in "Developing message flows that use WebSphere Adapters" on page 1106.
- Before you run the Adapter Connection wizard, gather the information that you need from the system administrator. You need information to connect to the EIS, and information to discover objects. For a list of the information that you might need to gather for each EIS, see the appropriate topic:
  - "Interacting with an SAP application" on page 1112
  - "Interacting with a Siebel application" on page 1122
  - "Interacting with a PeopleSoft application" on page 1126
  - "Interacting with a JD Edwards application" on page 1129

About this task

A message flow that uses one of the WebSphere Adapters requires the following resources:

- One or more message flows that contain one or more WebSphere Adapters nodes
- An XML Schema Definition (XSD) for each business object in the Enterprise Information System (EIS)
- An adapter component file for the WebSphere Adapter that is being used
- A library or application to contain these resources

The Adapter Connection wizard creates these resources automatically.

You can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the EIS by running the Adapter Connection - Iterative Discovery wizard. This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects. For more information, see "Enhancing existing adapters with newly discovered objects" on page 1120.

The following steps describe how to connect to an EIS.

Procedure

1. Click **File** > **New** > **Adapter Connection**.

   Alternatively, click **Quick Starts**, then click **Start from adapter connection**.

   The Adapter Connection wizard opens.
2. Follow the instructions in the wizard. To see a description of each field within the wizard, hold the mouse pointer over the field.

   Ensure that inbound and outbound SAP IDocs have different names if they are stored in the same library.

   When you have completed the steps in the wizard, the specified library contains a message type for each business object that is to be used.

   When the Adapter Connection wizard completes, the IBM App Connect Enterprise Toolkit displays a message that prompts you to drag the adapter component onto the message flow canvas.
3. Ensure that a message flow is open in the Message Flow editor so that the message flow canvas is available.
4. In the Application Development view, expand the folders beneath the library until you see the adapter component, which has a suffix of inadapter or outadapter.
5. Drag the adapter component onto the message flow canvas.

   The component appears as a message flow node.
6. Configure the node, as described in "Configuring a message flow node" on page 624.
7. When you have developed and saved your message flow, deploy it by following the instructions in "WebSphere Adapters resources" on page 2475.

What to do next

Using policies for WebSphere Adapters nodes

WebSphere Adapters nodes can get connection details from either the adapter component or a policy. By using policies, you can change the connection details for adapters without the need to redeploy the adapters. For more details about using policies with WebSphere Adapters nodes, see the appropriate topic:

- "Changing connection details for SAP adapters" on page 209

Configuring EIS connections to expire after a specified time
You can configure connections to SAP, Siebel, and PeopleSoft to expire after a specified time by using a policy.

About this task

You can use the `Connection idle timeout` property on a policy to control the number of connections to SAP, Siebel and PeopleSoft by closing connections that have not been used for a specified time.

Procedure

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select an appropriate policy type, such as **SAP Connection** (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).

2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the message flow node that you want to override.

   Specify the name of the policy project and the policy on the message flow node in the format *{policyProjectName}:PolicyName*.
3. Set the `Connection idle timeout` property on the policy to the number of seconds for which the connection can be idle before it is closed.

   The default value for this property is 0 (zero) seconds, indicating that no timeout occurs. Note that new connections to SAP are opened with different user IDs; therefore do not set this property to zero if you are using identity propagation.
4. Deploy this policy before you start the associated message flow.

Configuring WebSphere Adapters nodes for secondary adapters
You can deploy the resources that are required to support new methods in an Enterprise Information System (EIS), without affecting any resources that are already deployed, by using primary and secondary adapters. You must configure the WebSphere Adapters nodes in your message flows to use the secondary adapters.

Before you begin

- Read the concept topic, "WebSphere Adapters resources" on page 2475.
- Complete the steps in "Preparing your system to use WebSphere Adapters nodes" on page 1106.

About this task

The primary adapter for a WebSphere Adapters node contains its connection information and part of its interface; the secondary adapters contain the rest of the interface. The primary library contains message model metadata for the parts of the interfaces that are supported by the node; the secondary libraries define the rest of the model.

Complete the following steps to configure a message flow so that it can be extended by secondary adapters and libraries.

Procedure

1. To create a primary adapter and library, run the **Adapter Connection** wizard by following the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
2. When the wizard has finished, create a message flow that contains one or more WebSphere Adapters nodes.
3. In the Application Development view, expand the folders beneath the library until you see the adapter component, which has a suffix of `.inadapter` or `.outadapter`.
4. Drag the adapter component onto the WebSphere Adapters node in your message flow.
5. On the Basic properties tab of the WebSphere Adapters node, set the `Secondary adapter mode` to `All adapters in application`.
6. Save the message flow.

What to do next

Add the library to a BAR file, then deploy the BAR file to the integration server. For more information, see "Including WebSphere Adapters resources in a BAR file" on page 2475.

You have created a message flow that you can enhance in the future when you need to call new services or handle new event types in an EIS. For instructions about how to enhance your message flow, see the following topics:

- "Calling new services from a WebSphere Adapters request node without changing existing deployed resources" on page 1111

- "Handling new event types from a Websphere Adapters input node without changing existing deployed resources" on page 1112

Calling new services from a WebSphere Adapters request node without changing existing deployed resources
If your message flow acts as a gateway to an Enterprise Information System (EIS), you can use it to call new services that did not exist when you developed the flow. Therefore, if a new service is provided by the EIS, you do not have to modify and retest the message flow.

Before you begin

- Read the concept topic "WebSphere Adapters resources" on page 2475.
- Ensure that you have created and deployed a message flow that is configured to use secondary adapters, as described in "Configuring WebSphere Adapters nodes for secondary adapters" on page 1110.

About this task

To use iterative deployment, you can configure an SAPRequest, SiebelRequest, PeopleSoftRequest, or JDEdwardsRequest node to look for a specified operation in all relevant `.outadapter` components that are deployed to the integration server.

Procedure

1. Run the **Adapter Connection** wizard to create an `.outadapter` component and a library for the new EIS services.

   You do not need to rediscover existing services. For more information about running the Adapter Connection wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
2. Ensure that method names are unique across all primary and secondary adapters that are used by the request node. If they are not unique, edit them by clicking **Edit Operations** on the **Service Generation and Deployment Configuration** panel of the **Adapter Connection** wizard.

   The method names correspond to the Service Operation names, which are configured by the **Adapter Connection** wizard. In most cases, the names are based on the name of the service that is being discovered (for example, the BAPI name in SAP, or the business object and operation name in Siebel). However, in some cases, you must edit them to avoid a clash.

   You can set the method name dynamically in the message flow by setting the local environment field `$LocalEnvironment/Adapter/MethodName`.
3. Avoid duplicate method names by using user trace in the following way.
   a) Start user trace by following the instructions in "User trace example" on page 3030.
   b) Stop and restart the message flow.
   c) Read user trace. Message BIP3432 identifies which methods are already defined by currently deployed adapters.

   Alternatively, you can identify the methods that are defined by the adapter by looking at the `Default method` property of the request node. If you have a `.outadapter` component in your workspace, you can drop it onto a message flow to create a request node for the `.outadapter` component. The request node has that adapter set as its primary adapter, and the list of methods that are defined by that adapter are visible in the `Default method` property of the request node.
4. Ensure that the library that is created does not contain any types that share the same name and namespace of existing libraries. You can change the namespaces of the types on the **Adapter Connection** wizard by using the Business Object Namespace control.

   Use different namespaces for different libraries. The use of different namespaces is important when working with BAPIs because the BAPI return field typically has the same name for all BAPIs, and its type definition can change depending on the age of the BAPI.

Handling new event types from a Websphere Adapters input node without changing existing deployed resources

You can create an event handler to an Enterprise Information System (EIS) to handle new event types that did not exist when you first developed your message flow. Therefore, if a new event is provided by the EIS, you do not have to modify and retest the message flow.

Before you begin

- Read the concept topic "WebSphere Adapters resources" on page 2475.
- Ensure that you have created and deployed a message flow that is configured to use secondary adapters, as described in "Configuring WebSphere Adapters nodes for secondary adapters" on page 1110.

About this task

To implement an event handler that you can enhance iteratively with new event types (that is, deploy new event types to the event handler), configure an SAPInput node to use secondary adapters. The set of secondary adapters can be extended at operational time without affecting any existing adapters.

Procedure

1. Run the **Adapter Connection** wizard to create a new `.inadapter` component and a new message set for the new EIS events.

   You do not need to rediscover existing events. For more information about running the Adapter Connection wizard, see "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
2. Ensure that the method names are unique.

   The method names correspond to the Service Operation names. Therefore, when you are choosing or editing the Service Operation names, make sure that they do not clash with those names defined in other adapters that are being used by that node.
3. Avoid duplicate method names by using user trace in the following way.
   a) Start user trace by following the instructions in "User trace example" on page 3030.
   b) Stop and restart the message flow.
   c) Read user trace. Message BIP3432 identifies which methods are already defined by currently deployed adapters.
4. Ensure that the message set that is created does not contain any types that share the same name and namespace of existing message sets. You can change the namespaces of the types on the **Adapter Connection** wizard by using the Business Object Namespace control.

   Use different namespaces for different message sets. The use of different namespaces is important when working with BAPIs because the BAPI return field typically has the same name for all BAPIs, and its type definition can change depending on the age of the BAPI.

Interacting with an SAP application

To interact with an SAP application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to an SAP application, the SAP adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the SAP application, then deploy the relevant resources.

Procedure

1. To obtain software dependencies, follow the instructions in "Adding external software dependencies for SAP" on page 1113.

   If you require your message flow to use transactional processing (which is specified by default on the SAPInput node), ensure that IBM MQ is installed on the same computer as IBM App Connect Enterprise. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.
2. Optional: If you are using Application Link Enabling (ALE) processing, register a Remote Function Call (RFC) destination on the SAP server and configure the SAP server, as described in "Configuring the SAP server to work with the adapter" on page 1115.
3. Before you run the Adapter Connection wizard, gather the following information from your SAP administrator:

   - SAP system user name
   - SAP system password
   - SAP host name or IP address
   - SAP Client ID (for example, 001)
   - SAP system number (for example, 00)
   - Language code (for example, EN)

   You might also need to gather some specific information from your SAP administrator about the items that you are discovering. For example, if you are discovering IDocs, you need the appropriate information to complete the following fields:

   - IDoc name
   - Version
   - SAP outbound
   - Verbs (create, update, delete)
   - Partner number
   - Message type

   For more information, see SAP connection properties for the Adapter Connection wizard.
4. To connect to SAP by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
5. Develop your message flow to define the interaction with the SAP application.
6. Optional: To propagate security credentials to an SAP request, see "Propagating security credentials to Siebel and SAP requests" on page 1121.
7. Deploy the appropriate resources, as described in "Including WebSphere Adapters resources in a BAR file" on page 2475.
8. Optional: To enhance existing adapters with newly discovered objects, see "Enhancing existing adapters with newly discovered objects" on page 1120.

Adding external software dependencies for SAP
Before you can develop message flows that use WebSphere Adapters nodes, you must add prerequisite files to the runtime environment.

Before you begin

- If you require transactional support for SAP nodes, ensure that an IBM MQ client or server is installed on the same computer as the integration server. If you do not require the nodes in the flow to be transactional, you can set the **Transaction mode** property of the SAP node to No, and access to IBM MQ will not be required. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

- Ensure that you have the relevant prerequisite files for your SAP system. (For information about the versions of files that IBM App Connect Enterprise supports, see IBM App Connect Enterprise system requirements.)
  - `sapidoc3.jar`
  - `sapjco3.jar`
  - Windows On Windows:

    - `sapjco3.dll`
  - Linux UNIX On Linux and UNIX:

    - `libsapjco3.so`

  Note: You can download these files for your operating system from the external SAP Web site, SAP Support Portal, and save them to a directory, such as `C:\SAP_LIB`. (On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk.) You must have an SAPNet account to be able to access this Web site.
  - Windows On Windows, download the `.dll` files that come with the SAP JCo download.
  - Linux UNIX On Linux and UNIX, download the `.so` and `.o` files that come with the SAP JCo download.

Locating the SAP support files in the runtime environment

About this task

The integration server configuration file (`server.conf.yaml`) contains a section for connector providers, which defines the location of the required JAR files and native libraries, as you can see in the following example:

```
ConnectorProviders:
  SAPConnectorProvider:
    jarsURL: default_Path     # Set to the absolute path containing the SAP JCo JARs
    nativeLibs: default_Path  # Set to the absolute path containing the SAP JCo libraries
```

By default, the location is set to `default_Path`, which indicates that the JAR files and libraries can be found in the integration server's shared-classes directory.

To specify a location for the SAP prerequisite files, complete the following steps.

Procedure

- Linux Windows UNIX

  On Windows, Linux, and UNIX:

  a) Use a YAML editor to open the `server.conf.yaml` file, then find the `ConnectorProviders` section.

  For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

  b) Set the `jarsURL` property to the absolute path where the JAR files are stored.

  c) Set the `nativeLibs` property to the absolute path where the native libraries are stored.

  d) Save the `server.conf.yaml` file, then restart the integration server.

Locating the SAP support files through the IBM App Connect Enterprise Toolkit

About this task

To add the SAP prerequisite files through the IBM App Connect Enterprise Toolkit, take the following steps.

Procedure

- Windows

  On Windows: When you run the Adapter Connection wizard, you are prompted to specify the paths to the required libraries.

- Linux

  On Linux: Append the directory that contains the SAP libraries to the LD_LIBRARY_PATH environment variable:

  ```
  LD_LIBRARY_PATH=DIRECTORY_CONTAINING_SAP_LIBRARIES:${LD_LIBRARY_PATH}
  export LD_LIBRARY_PATH
  ```

  You must set this variable either for the whole system, or in the same shell or environment from which the IBM App Connect Enterprise Toolkit is launched.

What to do next

Configure the SAP system to work with the adapter

Configuring the SAP server to work with the adapter

Before you configure the WebSphere Adapter for SAP Software for Application Link Enabling (ALE) processing, you must register a Remote Function Call (RFC) destination on the SAP server, and configure a receiver port, logical system, distribution model, and partner profile on the SAP server. Ask your system administrator if these items have been configured.

Before you begin

Add the required external software dependencies for SAP.

About this task

Complete the following steps on the SAP server by using the SAP graphical user interface.

Procedure

1. Register an RFC program ID:
   a) Open transaction **SM59** (Display and Maintain RFC Destinations).
   b) Click **Create**.
   c) Type a name for the RFC destination.
   d) In the **Connection Type** field, select **T**.
   e) In the **Activation Type** field, select **Registered Server Program**.
   f) Type a Program ID.

      You use this program ID when you configure the adapter. This value indicates to the SAP gateway which RFC-enabled functions the program ID listens for.
   g) Enter a description in **Description 1**, such as `RFC for Test Sample`.
   h) Enter a description in **Description 2**, such as your name.
   i) Click **MDMP & Unicode**, and set the RFC destination to **Unicode** or **non-Unicode**, depending on the communication type of the target system.

      To avoid errors when you use multiple language settings, set the RFC destination to **Unicode**.
   j) Save your entry.
2. Set up a receiver port:
   a) Open transaction **WE21** (Ports in IDoc processing).
   b) Select **Transactional RFC**, click **Ports**, and click the **Create** icon.
   c) Type a name for the port and click **OK**.
   d) Type the name of the destination that you created in the previous task (or select it from the list).
   e) Save your entry.
3. Specify a logical system:
   a) Open transaction **BD54** (Change View Logical Systems).
   b) Click **New Entries**.
   c) Type a name for the logical system and click the **Save** icon.
   d) If you see the Prompts for Workbench request, click the **New Request** icon. Then enter a short description and click **Save**.
   e) Click the **Continue** icon.
4. Configure a distribution model:
   a) Open transaction **BD64** (Maintenance of Distribution Model).
   b) Click **Distribution Model** > **Switch processing model**.
   c) Click **Create model view**.
   d) Type a name for the model view and click the **Continue** icon.
   e) Select the distribution model that you created and click **Add message type**.
   f) For outbound processing, type the logical system name that you created in the previous task as **Sender**, and type the logical name of the SAP server as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
   g) Select the distribution model again and click **Add message type**.
   h) For inbound processing, type the logical name of the SAP server as **Sender**, and the logical system name that you created in the previous task as **Receiver**, then select a message type (for example, **MATMAS**) and click the **Continue** icon.
   i) Save your entry.

5. Set up a partner profile:
   a) Open transaction **WE20** (Partner Profiles).
   b) Click the **Create** icon.
   c) Type the name of the logical system that you created in the earlier task and, for **Partner Type**, select **LS**.
   d) For **Post Processing: permitted agent**, type US and your user ID.
   e) Click the **Save** icon.
   f) In the Outbound parameters section, click the **Create outbound parameter** icon.
   g) In the **Outbound parameters** window, type a message type (for example, MATMAS05), select the receiver port that you created in the earlier task, and select **Transfer IDoc immed**.
   h) Click the **Save** icon.
   i) Press **F3** to return to the Partner Profiles view.
   j) In the Inbound parameters section, click the **Create inbound parameter** icon.
   k) In the **Inbound parameters** window, type a message type (for example, MATMAS), and a process code (for example, MATM).
   l) Click the **Save** icon.
   m) Press **F3** to return to the Partner Profiles view.
   n) In the Inbound parameters section, click the **Create inbound parameter** icon.
   o) In the **Inbound parameters** window, type the following values: ALEAUD for **Message Type**, and AUD1 for **Process Code**.
   p) Click the **Save** icon.
   q) Press **F3** to return to the Partner Profiles view.
   r) Click the **Save** icon.

What to do next

connect to an EIS using the Adapter Connection wizard.

Changing connection details for SAP adapters

SAP nodes can get SAP connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read “WebSphere Adapters nodes” on page 1050 and “Overriding properties at run time with policies” on page 324 for background information.

About this task

Use the SAP Connection policy to change connection details for an SAP adapter. The SAP node reads all connection properties from the adapter component that it is configured to use. If a SAP Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the SAP policy are described in SAP Connection policy (SAPConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **SAP Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the SAPInput or SAPRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see SAP Connection policy (SAPConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Setting up SAP for high availability
You can configure IBM App Connect Enterprise to withstand software or hardware failures when working with SAP by moving the transaction ID (TID) store to a client queue manager that can be shared between two integration nodes. To avoid a single point of failure, make this queue manager an IBM MQ high availability, multi-instance queue manager.

Before you begin

- Read the concept topic about "SAP high availability" on page 1070.
- Ensure that either an IBM MQ client or an IBM MQ server is installed on the same computer as the integration node. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

About this task

The following topic describes how to set up an IBM MQ shared queue:

Procedure

- "Setting up a shared queue for the SAP adapter event store" on page 1118

Setting up a shared queue for the SAP adapter event store
To achieve high availability when processing SAP messages on distributed systems, you can set up a shared queue for the SAP adapter event store.

About this task

On distributed systems, you can configure the integration node to use a client connection to a remote IBM MQ queue manager to persist the transaction ID (TID) store for SAP transactional RFC (tRFC) data. By using this configuration, two adapters that are deployed to two integration nodes can share the same TID, and can therefore operate as a single RFC server. This configuration is essential if the adapters have been configured with the same RFC Program ID. For important information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

Before the integration node can use a remote queue manager as the TID store, you must complete some administration tasks on that queue manager.

1. You must first create the queue for the integration node to use as the TID store.

2. Then you must define two channels for the integration node to use to connect to that queue manager: define the server channel on the queue manager, and define a client channel in a file, which you must make available to the integration node.

   You can use the Run WebSphere MQ Commands (runmqsc) command or IBM MQ Explorer to create this client channel definition file on any queue manager. After you have created the file, you must move it from the queue manager to a file system that is accessible to the integration node.

To set up a shared queue for the SAP adapter event store, complete the following steps.

Procedure

1. Create the queue SYSTEM.BROKER.ADAPTER.PROCESSED.

   You can create the queue either by using IBM MQ Explorer or by running the following command in IBM MQ **runmqsc**:

   ```
   DEFINE QLOCAL('SYSTEM.BROKER.ADAPTER.PROCESSED')
   ```

2. Create the server channel.

   a) On the computer that hosts the shared queue manager, create a server channel by running the IBM MQ **runmqsc** command and entering the following parameters, where *WSADAPTERS.SAP* is an example of the channel name.

   ```
   DEFINE CHANNEL ('WSADAPTERS.SAP') CHLTYPE (SVRCONN) TRPTYPE(TCP)
   ```

   b) After you have created the channel, start it by using the following command in **runmqsc**:

   ```
   START CHANNEL ('WSADAPTERS.SAP')
   ```

3. Create the client definition file.

   On any queue manager, create a client definition file by running the **runmqsc** command and entering the following parameters, where the following criteria apply:

   - *WSADAPTERS.SAP* is an example of the channel name.
   - *myhost.ibm.com* is an example of the host name of the computer where the queue manager is running. This queue manager is the one on which you created the queue in step 1 and the server channel in step 2. This name is used by the integration node to connect to the queue manager, therefore do not use *localhost*.
   - *1414* is an example of the port number where the listener is running on that queue manager.
   - *QMGR* is an example of the queue manager name.
   - You can use any name for the server and client channels, but they must match.
   - Set the CONNAME parameter to the host name or IP address of the computer that hosts the shared queue manager, and the port number of the MQ Listener that is running on that computer. 1414 is the default listener port.

   ```
   DEFINE CHANNEL ('WSADAPTERS.SAP') CHLTYPE (CLNTCONN) CONNAME('myhost.ibm.com(1414)')
    TRPTYPE(TCP) QMNAME(QMGR)
   ```

   If the queue manager is running in multi-instance mode, define two channels, one for each instance of the queue manager

4. Move the client definition file to a file system that is accessible by the integration node.

   - Linux UNIX On Linux and UNIX, you can find this file at `/var/mqm/qmgrs/QMGR_NAME/@ipcc/AMQCLCHL.TAB`.
   - Windows On Windows, you can find this file at `C:\Program Files\IBM\IBM MQ\Qmgrs\QMGR_NAME\@ipcc\AMQCLCHL.TAB`.

5. Configure the integration node.
   a) Create an SAP Connection policy for the `.inadapter` component and ensure that the name of the policy matches the name of the adapter component (see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327).
   b) Set the `Shared TID store client definition file` property of the policy to the path of the file that you created in step 3 and moved in step 4.
   c) Set the `Shared TID store queue manager` parameter to the name of the queue manager for which you created the queue in step 1 and the server channel in step 2.

   For more information about these parameters, see SAP Connection policy (SAPConnection).
6. Deploy the SAP Connection policy.
7. Verify the setup.

   After you have completed these steps, you can verify that the integration node is using the queue on the remote queue manager by inspecting user trace. If the setup is successful, message BIP3470 is issued, specifying which queue manager the integration node is using as the TID store.

Enhancing existing adapters with newly discovered objects
In WebSphere Message Broker Version 8.0, you can take an adapter component that was created by using the Adapter Connection wizard, and update it with newly discovered objects from the Enterprise Information System (EIS). This facility is known as *iterative discovery*. You can either add the new objects without modifying existing objects, or replace existing objects.

Before you begin

The following instructions assume that you have already run the Adapter Connection wizard to discover a set of objects, as described in “Connecting to an EIS by using the Adapter Connection wizard” on page 1108. You can run iterative discovery only on projects that were created in WebSphere Message Broker Version 7.0 or later versions.

About this task

Procedure

1. From the Application Development view, expand the appropriate message set project folders until you see the .inadapter or .outadapter component.
2. Right-click the adapter component, then click **Iterative discovery**.

   The Adapter Connection - Iterative Discovery wizard opens. The fields in the wizard are populated with the values that you specified when you ran it previously. You can modify these values if anything has changed, such as a password.
3. On the **Find and Discover Services** page, the objects that you previously discovered and selected for import are shown in the **Objects to be imported** pane. You can discover and select new objects to import from the **Objects discovered by query** pane.

4. On the final page of the Adapter Connection - Iterative Discovery wizard, choose from the following options.
   - To add only the new objects that you discovered, select **Add XSDs for new objects and replace .wsdl, .import/.export**. When you click **Finish**, the following results occur:
     – XSD files for the additional objects are added to the adapter component. These files are shown in the **New files** pane.
     – Updated .wsdl and .import or .export files replace the files already in the adapter component.
     – The XSD files that are being added to the adapter component are also imported into the message set that is associated with the adapter component.

     If you have discovered objects by using the SAP Wrapper or SAP Work Unit options, the XSD files that correspond to the SAP wrapper objects are replaced in the adapter component. These XSD files are also imported into the message set, replacing any files that have the same name. This behavior is because the wrapper objects contain references to the new objects.
   - To replace all existing objects with the objects that you have discovered, select **Replace the contents of .in/.outAdapter with the newly discovered files**. When you click **Finish**, the following results occur:
     – The contents of the .inadapter or .outadapter file in the adapter component are replaced.
     – All discovered XSD files are imported into the message set, replacing any files that have the same name.

   Note:

   Select the **Replace the contents of .in/.outAdapter with the newly discovered files** option if you have changed any of the configuration options during rediscovery, or if the interfaces or data structure of objects that you are discovering have changed since the previous discovery.

   If you have discovered objects by using the BAPI result set option, the **Add XSDs for new objects and replace .wsdl, .import/.export** option is not available and you see the message: `Incremental discovery is not applicable for the selected option`.

   For options that are used in rediscovering SAP objects, and the corresponding restrictions, see SAP options for rediscovery.
5. Click **Finish**.

Propagating security credentials to Siebel and SAP requests
The SAPRequest and SiebelRequest nodes can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request. The nodes must use the `Propagate` property on the security profile that is defined on the node to propagate the security credentials.

About this task

If an SAPRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to SAP. Similarly, if a SiebelRequest node is configured with a security profile, it extracts tokens from the input message at run time, and propagates an identity to Siebel.

Procedure

To propagate an identity to be used for the Siebel or SAP request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the SAPRequest node, or create a security profile, by following the instructions in “Creating a security profile” on page 2584.
2. Use the BAR editor to select a security profile for the SAPRequest node that has identity propagation enabled.

   For detailed instructions, see “Configuring a message flow for identity propagation” on page 2613.

Interacting with a Siebel application
To interact with a Siebel application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to a Siebel application, the Siebel adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the Siebel application, then deploy the relevant resources.

Procedure

1. To configure the Siebel application to work with the adapter, create an event table and a Siebel business object, as described in "Configuring the Siebel application to work with the adapter" on page 1122.
2. Before you run the Adapter Connection wizard, gather the following information from your Siebel administrator:
   - Siebel user name
   - Siebel password
   - Siebel host name or IP address
   - Language code

   For more information, see Siebel connection properties for the Adapter Connection wizard.
3. To connect to Siebel by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
4. Develop your message flow to define the interaction with the Siebel application.
5. Optional: To propagate security credentials to a Siebel request, see "Propagating security credentials to Siebel and SAP requests" on page 1121.
6. Deploy the appropriate resources, as described in "Including WebSphere Adapters resources in a BAR file" on page 2475.

Configuring the Siebel application to work with the adapter
To configure the Siebel application, create an event table and a Siebel business object.

Before you begin

1. Before you configure the Siebel application to work with WebSphere Adapter for Siebel Business Applications, you must create a user name and password so that the Adapter Connection wizard can connect to Siebel Business Applications to perform outbound operations, and retrieve Siebel business objects and services.

   You perform this task on the Siebel server, therefore ensure that you are familiar with the Siebel tools that are required to complete it. For information about using Siebel tools, refer to the Siebel tools documentation.

   To open Siebel Sales Enterprise on your local database, you must have administrative privileges.

About this task

To configure the Siebel application, you must create an event table and a Siebel business object. IBM App Connect Enterprise contains resources that help you to create the event components and triggers. This topic describes how to use those resources. You can also create the event table and Siebel business object manually; for more information, see "Creating the event store manually" on page 1123.

Procedure

1. Locate the Scripts folder at *install_dir*/tools/ResourceAdapters/Siebel_*version*/Scripts.

   The Scripts folder contains two folders: Siebel7.x.x and Siebel8.0. Each version has an Event_pkg folder, which contains a `.sif` file and a number of `.js` scripts. You use the `.sif` file to create the event components; it can add business objects, views, and all other Siebel objects to the Siebel repository. The `.js` scripts help you to create Siebel triggers.
2. To use the `.sif` file:
   a) Open Siebel tools and click **Tools** > **Import**.
   b) Import the `.sif` file.
   c) Merge the differences between the `.sif` file and the Siebel repository.
   d) Recompile the repository into a Siebel repository file (`.srf` file).
3. Use the `.js` scripts to create Siebel triggers.

   The provided samples show how to create entries in the inbound table when new Account objects are created.

Creating the event store manually
To configure the Siebel application, create an event table and a Siebel business object.

About this task

"Configuring the Siebel application to work with the adapter" on page 1122 describes how to use the samples that are supplied with WebSphere IBM Integration to configure the Siebel application. This topic describes how to create the event store manually.

The following steps describe how to create the event store to be used for inbound operations in the Siebel application. You can substitute all references to Siebel Sales Enterprise with the name of the Siebel application that you are using.

Procedure

1. Create a project called IBM, and lock the project with Siebel tools.
2. Using the object wizard, create an event table called CX_IBM_EVENT in which to store the events.
   a) In the event table, create the columns that are shown in the following table.

Column Name	Type	Length	Data Type	Required	Nullable	Status
DESCRIPTION	Data (public)	255	Varchar	No	Yes	Active
EVENT_ID	Data (public)	30	Varchar	Yes	No	Active
EVENT_TYPE	Data (public)	20	Varchar	Yes	No	Active
OBJECT_KEY	Data (public)	255	Varchar	Yes	No	Active
OBJECT_NAME	Data (public)	255	Varchar	Yes	No	Active
PRIORITY	Data (public)	10	Varchar	No	Yes	Active
STATUS	Data (public)	20	Varchar	Yes	No	Active

Column Name	Type	Length	Data Type	Required	Nullable	Status
XID	Data (public)	255	Varchar	Yes	No	Active

   b) Create a new business component called IBM Event.
   c) Create a new time stamp called Field Event, and map it to the CREATED column from CX_IBM_EVENT. Make the Type of this field DTYPE_UTCDATETIME.
   d) Create a new business object called IBM Event.
   e) Associate the IBM event business component to the IBM Event business object.
   f) Create an applet called IBM Event List Applet, and base it on the IBM Event business component that you have created.
   g) Create a view called IBM Event List View, and base it on the IBM Event business object that you have created.
   h) Create a screen called IBM Event Screen, and associate it with the IBM Event List View in the Siebel tools.
3. Create a page tab.
   a) Click **Start Application** > **Siebel Sales Enterprise**.
   b) Right-click the **Page** tab, and click **New Record**.
   c) Specify IBM Event as the screen name, and IBM Event for the **Text - String Override** field.
   d) Leave the **Inactive** field blank.
4. Create a new business object called Schema Version for your IBM project and associate it with the Schema Version business component.
   a) Apply the physical schema for the new tables to your local database by querying for the new table, CX_IBM_EVENT_Q and selecting the current query to create a physical schema. Leave the table space and index space blank.
   b) Click **Activate** to activate the new schema.
5. Add or modify the Siebel VB or e-scripts for the business component that corresponds to the business objects that are used at your site. Siebel scripts trigger event notification for business objects. Samples are located in the Samples folder in your adapter installation.
6. Create a new Siebel repository file by compiling the updated and locked projects on your local database. The new repository file has an extension of `.srf`.
7. Create and populate a new responsibility.
   a) Open Siebel Sales Enterprise on your local database.
   b) Create a new responsibility called IBM Responsibility for IBM Event List View.
   c) Add the employees or teams who are responsible for reviewing events to the newly created IBM Responsibility.
   d) Create a user name called IBMCONN (or another user name to be used by the adapter later). Add the user name to the newly created IBM Responsibility and also to the Administrative Responsibility.
8. Test the application in your local environment to ensure that you can see the IBM Event List View. An event is generated in the view after you create a record in the supported object. As part of the test, create a new Account business component instance in Siebel. Confirm that a new Account event is shown in the IBM Event List View (assuming that you have added the e-script trigger to the Account business component). If a new Account event is not displayed in the view, check for an error and fix it. For more information on the errors that might be generated, check either the Siebel support site or Siebel documentation.
9. When the test that you perform in Step 8 is successful, add your new and updated projects to your development server.
10. Activate the new table in the development server.
11. Compile a new Siebel repository (`.srf`) file on the server.

12. Back up the original repository file on the server.
13. Stop the Siebel server and replace the original repository file with the newly created one.
14. Restart the Siebel server.

Changing connection details for Siebel adapters
Siebel nodes can get Siebel connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the Siebel Connection policy to change connection details for a Siebel adapter. The Siebel node reads all connection properties from the adapter component that it is configured to use. If a Siebel Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the Siebel policy are described in Siebel Connection policy (SiebelConnection).

You can also connect to different versions of Siebel by creating an EIS Siebel Providers policy and setting the location of the appropriate library files (see "Connecting to different versions of Siebel" on page 1125).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **Siebel Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the SiebelInput or SiebelRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see Siebel Connection policy (SiebelConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Connecting to different versions of Siebel
You can access multiple versions of Siebel from a single integration node by using different versions of the client libraries.

About this task

You can connect to different versions of Siebel by configuring integration servers to use different library files. You can then connect to different versions of a Siebel server from two different integration servers.

The integration server configuration file (`server.conf.yaml`) contains a section for connector providers, which defines the location of the required JAR files and native libraries, as you can see in the following example:

```
ConnectorProviders:
  SiebelConnectorProvider:
    jarsURL: default_Path      # Set to the absolute path containing the Siebel JAR files
    siebelPropertiesURL: ''    # Set to the absolute path containing the Siebel properties file
    nativeLibs: default_Path   # Set to the absolute path containing the Siebel libraries
```

By default, the location is set to `default_Path`, which indicates that the JAR files and libraries can be found in the integration server's shared-classes directory.

Procedure

To configure integration servers with the locations of required JAR files and libraries, complete the following steps:

1. Use a YAML editor to open the `server.conf.yaml` file, then find the `ConnectorProviders` section.

   For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
2. For the Siebel connector provider, set the `jarsURL` property to the absolute path where the JAR files are stored.
3. Set the `nativeLibs` property to the absolute path where the native libraries are stored.
4. Save the `server.conf.yaml` file.
5. Repeat these steps to specify the file locations for an alternative version of the Siebel server on a second integration server.
6. Restart the integration servers.
7. Deploy the message flows that contain Siebel nodes to the appropriate integration server.

Interacting with a PeopleSoft application

To interact with a PeopleSoft application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

About this task

To connect to a PeopleSoft application, the PeopleSoft adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the PeopleSoft application, then deploy the relevant resources.

Procedure

1. To configure the PeopleSoft application to work with the adapter, create a custom event project, as described in "Creating a custom event project in PeopleTools" on page 1127.
2. Before you run the Adapter Connection wizard, gather the following information from your PeopleSoft administrator:

   - PeopleSoft user name
   - PeopleSoft password
   - PeopleSoft host name or IP address
   - Port number (for example, 9000)
   - Language code (for example, ENG)

   For more information, see PeopleSoft connection properties for the Adapter Connection wizard.
3. To connect to PeopleSoft by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.

4. Develop your message flow to define the interaction with the PeopleSoft application.
5. Deploy the appropriate resources, as described in "Including WebSphere Adapters resources in a BAR file" on page 2475.

Creating a custom event project in PeopleTools
The WebSphere Adapter requires an event project in PeopleSoft to perform asynchronous inbound event processing. Use PeopleTools to create the custom event project.

About this task

If your environment requires inbound event support, you must use a custom event project in PeopleSoft. A sample event project, IBM_EVENT_V600, is provided with the adapter. You can modify and use and the sample project, or you can create your own project by using PeopleTools. If you create your own project, make sure that you complete the following steps.

Procedure

1. Use PeopleTools Application Designer to create and name a new project.
2. Create the fields for the new project as described in the following table:

Field name	Field description
IBM_EVENT_ID	A numeric value that is retrieved from IBM_FETCH_ID record. This value is a unique ID for the event.
IBM_OBJECT_NAME	The name of the corresponding business graph.
IBM_OBJECT_KEYS	The get key property names in the Component Interface, followed by the key values in name-value pairs. This information is used for the component's retrieval from the EIS.
IBM_EVENT_STATUS	If the event is ready to be polled, the status is set to 0 and the IBMPublishEvent function is called.
IBM_OBJECT_VERB	The verb that is set on the business object graph that contains the retrieved business object.
IBM_EVENT_DTTM	The date on which the event is created. For a future dated event, this is the effective date.
IBM_NEXT_EVENT_ID	The field that has the latest event ID under the record IBM_FETCH_ ID. This field is incremented for each event that is added to the IBM_EVENT_TBL, and it populates the IBM_EVENT_ID field in that table.
IBM_XID	The transaction ID that is needed to provide assured event delivery.

3. Create a record named IBM_EVENT_TBL and add to it all the fields that you have just created, except IBM_NEXT_EVENT_ID.
4. Create a record named IBM_FETCH_ID and add to it only the IBM_NEXT_EVENT_ID field.
5. Open the IBM_FETCH_ID record, select the IBM_NEXT_EVENT_ID field, view the PeopleCode, and select **fieldformula.**
6. Copy the PeopleCode for a custom event project from PeopleCode for a custom event project to the project that you are creating.
7. Create a page under your project that contains the fields of the IBM_EVENT_TBL record at level 0. The page can have any name.

8. Create a component under your project that contains the page that you have just created. The component can have any name.
9. Create a Component Interface against this component and give it any name. Confirm that you want to default the properties that are based on the underlying component definition.
10. Build the entire project, selecting all create options.
11. Test and confirm that the Component Interface works, by using the Component Interface tester.
12. Generate the Java APIs for the Component Interface, then add the generated classes to the adapter classpath.

    For complete information about building a PeopleTools project and testing the PeopleSoft Component Interface, refer to PeopleSoft documentation.

Changing connection details for PeopleSoft adapters
PeopleSoft nodes can get PeopleSoft connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the PeopleSoft Connection policy to change connection details for a PeopleSoft adapter. The PeopleSoft node reads all connection properties from the adapter component that it is configured to use. If a PeopleSoft Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the PeopleSoft policy are described in PeopleSoft Connection policy (PeopleSoftConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **PeopleSoft Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the PeopleSoftInput or PeopleSoftRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see PeopleSoft Connection policy (PeopleSoftConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Interacting with a JD Edwards application
To interact with a JD Edwards application, obtain external software dependencies, run the Adapter Connection wizard, develop a message flow, then deploy the relevant resources.

Before you begin

Gather the following information from your JD Edwards administrator; you will need this information when you run the Adapter Connection wizard.

- JD Edwards EnterpriseOne environment name
- JD Edwards user name
- JD Edwards password
- The role that is associated with the user name

To find objects in the JD Edwards system, you use the Adapter Connection wizard to run a query to discover business functions or XML lists. Ask your JD Edwards administrator about the available business function libraries or XML list tables. To complete the wizard, you need to know the table type and the attributes to specify in the query. For more information, see JD Edwards connection properties for the Adapter Connection wizard.

You need the following information to complete the system connection information in the Adapter Connection wizard. You can find this information in the `jdbj.ini` file that is included with the external JD Edwards resource files.

```
[JDBj-BOOTSTRAP SESSION]
user=user
password=***
environment=JDEenv
role=*ALL
```

Connecting to a DB2 database

If you are connecting to a DB2 database, the JDBC driver file that is required for DB2 can be found in the JDBC driver files table. The `jdbj.ini` file contains the following example entries, which have been configured by the JD Edwards administrator:

```
[JDBj-BOOTSTRAP DATA SOURCE]

databaseType=W

[JDBj-JDBC DRIVERS]

UDB=COM.ibm.db2.jdbc.app.DB2Driver
```

Connecting to an Oracle database

If you are Connecting to an Oracle database, the JDBC driver files that are required for Oracle can be found in the JDBC driver files table. The `jdbj.ini` file contains the following example entries, which have been configured by the JD Edwards administrator:

```
[JDBj-JDBC DRIVERS]
ORACLE=oracle.jdbc.driver.OracleDriver

[JDBj-ORACLE]
tns=tnsnames.ora
```

About this task

To connect to a JD Edwards application, the JD Edwards adapter requires certain files and libraries. You must store these files so that they are accessible to the Adapter Connection wizard. The wizard creates various resources, such as an adapter component and message flow. After you have completed the wizard, you can develop a message flow to define the interaction with the JD Edwards application, then deploy the relevant resources.

Procedure

1. To connect to JD Edwards by using the Adapter Connection wizard, and create a message flow, follow the instructions in "Connecting to an EIS by using the Adapter Connection wizard" on page 1108.
2. Develop your message flow to define the interaction with the JD Edwards application.
3. Deploy the appropriate resources, as described in "Including WebSphere Adapters resources in a BAR file" on page 2475.

Changing connection details for JD Edwards adapters
JDEdwards nodes can get JD Edwards EnterpriseOne connection details from either the adapter component or a policy. By using a policy, you can change the connection details for adapters without the need to redeploy the adapters. To pick up new values when a policy is created or modified, you must restart the integration server where the adapter is deployed.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050 and "Overriding properties at run time with policies" on page 324 for background information.

About this task

Use the JDEdwards Connection policy to change connection details for a JD Edwards adapter. The JD Edwards node reads all connection properties from the adapter component that it is configured to use. If a JDEdwards Connection policy exists with the same name as the node's **`Primary adapter component`** property, the node uses the values that are defined in that policy to override the corresponding properties from the adapter. If any properties on the policy are set to an empty string, the values that are configured in the `.inadapter` or `.outadapter` file are used. The properties of the JD Edwards policy are described in JDEdwards Connection policy (JDEdwardsConnection).

Procedure

To use a policy to change connection details at run time, complete the following steps:

1. Use the Policy editor in the IBM App Connect Enterprise Toolkit to create a policy and select **JD Edwards Connection** as the policy type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Ensure that the name of the policy is the same as the name that is specified by the `Primary adapter component` property on the JDEdwardsInput or JDEdwardsRequest node (without the `.inadapter` or `.outadapter` suffix).
3. Set the appropriate properties in the policy for the settings that you want to use at run time.

   The default value for most properties in the policy is an empty string, which indicates that behavior is controlled by the `.inadapter` or `.outadapter` file for those properties. For more information, see JDEdwards Connection policy (JDEdwardsConnection).
4. Deploy this policy in the default policy project for the integration server before you start the associated message flow.

   For information about configuring a default policy project, see "Configuring a default policy project" on page 329.

Working with databases

Create and configure databases to use with your message flows.

About this task

- "Databases overview" on page 1131
- "Accessing databases from message flows" on page 1134
- "Accessing databases from ESQL" on page 1135

- “Event-based database integration” on page 1136
- “Configuring a DatabaseInput node” on page 1140

Databases overview

The integration server can manipulate business data that is stored in databases. If you have databases, you must configure them before you can access them from your message flow.

IBM App Connect Enterprise supports the databases that are listed in IBM App Connect Enterprise system requirements. If you configure your message flows to access databases, you cannot access some of the data types that are supported by these databases. The supported data types are defined in Data types of values from external databases.

Additional local and remote database managers might also be supported for your computer. For more information, see IBM App Connect Enterprise system requirements.

You must set up connections to the databases so that the integration node can access the databases on behalf of its deployed message flows. Both ODBC and JDBC connections are supported; some restrictions apply on some platforms, as described in the topics in this section. ODBC drivers are supplied and installed with IBM App Connect Enterprise. JDBC drivers are not supplied by IBM App Connect Enterprise; you must obtain these files from your database vendor.

For information about connections to databases, see “Database connections” on page 1131.

Database connections

Databases contain business data that is written and accessed by deployed message flows. You must create connections from the integration node to the database by using ODBC or JDBC.

A database connection is a configuration file where you specify a database's physical details such as database type and version, and parameters to enable a JDBC connection from IBM App Connect Enterprise Toolkit to the database.

ODBC connections to databases are managed internally by the integration node, and therefore any configurable connection pooling options that are available on the ODBC driver should not be used.

The integration node requires a database connection for each data source name (DSN) that is referenced in the message flow, even if different DSNs resolve to the same physical database. If the message flow is operating in coordinated mode, then a separate XA connection to each DSN participating in the globally coordinated flow is also required.

The number of connections to a database that an integration node requires depends on the actions of the message flows that access the database. For each message flow thread, an integration node that accesses a database makes one connection for each data source name (DSN). If a different node on the same thread uses the same DSN, the same connection is used, unless a different transaction mode is used, in which case another connection is required. For further information about transactions, see Database connections for coordinated message flows.

Normally, the integration node makes the connections when it needs to use them in the message flow. However, in the case of connections to ESQL-based data sources, you can make the initial connection before a flow receives a message, and so remove any connection latency from the message processing. Set this option by selecting **`Connect before flow starts`** property on the data source node.

If the message flow contains a DatabaseInput node, at least one database connection remains open while the message flow is running.

On Linux, UNIX, and Windows systems, database connections are released under the following circumstances:

Database connections that have been started with the **`Connect before flow starts`** option selected are not released when the message flow becomes idle. These types of connections are only released in the following circumstances:

- An error occurs, while accessing the database, that requires a new connection to be made.
- The message flow is stopped.

- The integration node is stopped.

All other database connections remain open until one of the following events occurs:

- An error occurs, while accessing the database, that requires a new connection to be made.
- The message flow has been idle for 1 minute.
- The message flow is stopped.
- The integration node is stopped.

z/OS On z/OS, database connections are released if the database has not been accessed for 1 minute.

To change the default time of 1 minute after which a database connection for an idle message flow is released, use the following command:

```
mqsichangeproperties integration_node -e integration_server -o ComIbmDatabaseConnectionManager -
n maxConnectionAge -v newValue
```

or the following command, to change the default time for all integration servers:

```
mqsichangeproperties integration_node -o ComIbmDatabaseConnectionManager -n maxConnectionAge -
v newValue
```

where *maxConnectionAge* is specified in seconds. If *maxConnectionAge* is set to option -1, database connections are never released until the integration server or integration node is stopped.

If you are using DB2 for your database, the default action is to limit the number of concurrent connections to a database to the value of the **maxappls** configuration parameter. The default for **maxappls** is 40. If you believe that the connections that the integration node might require exceeds the value for **maxappls**, increase this parameter and the associated **maxagents** parameter to new values based on your calculations.

For z/OS, the number of connections does not change when you use ODBC CAF (Call Attachment Facility) connections or RRSAF (Recoverable Resource Services Attachment Facility).

Use the **maxConnectionUseCount** configuration parameter to release an ODBC connection to the database after a specified number of transactions on a thread is reached. To activate this function, set **maxConnectionUseCount** to a value that is greater than zero.

maxConnectionUseCount is monitored on a per thread basis. For example, if you have additional instances of a message flow, meaning that you have multiple threads running in parallel, the connection to the database is released only on those threads that reach the value specified in **maxConnectionUseCount**.

For an integration server that is managed by an integration node, you can set the value of **maxConnectionUseCount** by using the **mqsichangeproperties** command. For example, to reset the ODBC connection after every 100 transactions, including commit and rollback processes, for each datasource connection on a thread, issue the following command:

```
mqsichangeproperties integration_node -e integration_server -o ComIbmDatabaseConnectionManager -
n maxConnectionUseCount -v 100
```

Restart the integration server for the update to take effect.

Use the **mqsireportproperties** command to discover the current number of completed transactions on each datasource connection. This value is reported in the **transactionCount** field. For example:

```
mqsireportproperties integration_node -e integration_server -o ComIbmDatabaseConnectionManager -
r


ComIbmDatabaseConnectionManager
  name='DatabaseConnectionManager'
  identifier='ComIbmDatabaseConnectionManager'
  type='resourceManager'
  enableODBCTraceBridge='true'
```

```
  expireXAConnections='false'
  maxConnectionAge='60'
  maxConnectionUseCount='100'
  maxStatementAge='600'
  statementCacheSize='40'
  useDefaultSchemaForStoredProcedures='true'
  active
    activityLogUri='/apiv2/servers/default/resource-managers/activity-log-manager?
log_type=RM&resource_manager=ODBC'
    expireXAConnections='false'
    maxConnectionAge='60'
    serverRestartRequired='false'
    detailed
      databaseContexts
        databaseContext
          threadNumber='27546'
          ODBC_Version_3
            envHandle='139789946390048'
            dataSources
              dataSource
                connectionAge='34'
                coordinated='false'
                name='TESTDB'
                state='2'
                timeoutEnabled='true'
                transactionCount='2'
                managedStatements
                  managedStatement
                    createTime='2020-08-11 04:50:15.285704 (1599799815)'
                    executeCount='2'
                    lastExecuteTime='2020-08-11 04:50:40.404019 (1599799840)'
                    text='INSERT INTO TEST VALUES (?)'
                statements
```

For an independent integration server, you can update the value of **maxConnectionUseCount** in the **ResourceManagers** section of the `server.conf.yaml` file. For example:

```
ResourceManagers:
  DatabaseConnectionManager:
     maxConnectionUseCount: '100'
```

Restart the integration server for the update to take effect.

When the release function that is activated when **maxConnectionUseCount** is set to a value greater than zero, connection release is prompted when whichever of the values in **maxConnectionAge** or **maxConnectionUseCount** is reached first. To serve as an example, consider a scenario where **maxConnectionUseCount** is set to 100, and **maxConnectionAge** is set to 60. In such a scenario:

- If 100 transactions are processed within 30 seconds of the creation of an ODBC connection, the ODBC connection is released immediately after the number of transactions reaches 100. A new ODBC connection will be established on the next message.
- If 10, say, transactions are processed within 30 seconds of the creation of an ODBC connection, and for next 60 seconds, or more, no messages are processed on this ODBC connection, the value specified in **maxConnectionAge** is reached and the connection is released.
- If 10, say, transactions are processed every 30 seconds, and the connection is never idle for more than 60 seconds, the ODBC connection is released after the total transaction count reaches 100, which, in this example, is after 300 seconds.

Note: The **connectionAge** field that is returned as part of the output from the **mqsireportproperties** command is not a configurable property; it is a runtime metric that indicates the age of the connection. There is no relationship between **connectionAge** and **maxConnectionAge**. The value in **connectionAge** can exceed the value in **maxConnectionAge** as seen in the example scenario description above where **connectionAge** would be 300 and **maxConnectionAge** is 60.

If you are using another database, check the database documentation for information about connections and the limits or restrictions that might apply.

When a message flow is idle, the integration server periodically releases database connections that have not been started with the **Connect before flow starts** option. Therefore, connections held by the integration node reflect its current use of these resources. This situation allows the integration node

to respond when a database quiesces, if the database manager supports quiescing. Not all databases support the quiesce function, and not all databases quiesce in the same way. Check your database documentation for information about database quiescing.

Accessing databases from message flows

Create and configure message flows to access databases.

Before you begin

Complete the following tasks:

- "Creating a message flow" on page 574
- "Working with databases" on page 1130

Read the following concept topic:

- "Message flow nodes" on page 484

Check which databases are supported on which platform, and if any restrictions apply:

- IBM App Connect Enterprise system requirements

About this task

You can access a database from a message flow in two ways:

- You can design a message flow that responds to events generated by the database.
- After a flow has already started, you can access the database to read or update information in it. Information from the database can be used to enhance or influence the operation of the message flow.

You can access a database from a message flow by using the following nodes:

- Compute
- Database
- DatabaseInput
- DatabaseRetrieve
- DatabaseRoute
- Filter
- JavaCompute
- Mapping

For more details about these nodes, and how to configure them in message flows, see Built-in nodes.

To access a database from a message flow:

Procedure

1. Identify the database that you want to access.
2. Define a connection to the data source name (DSN) to enable a connection to the database, if one does not exist:
   - Define a JDBC connection if you want to interact with a database directly from a Java application, or from a Mapping node. You can code Java in both a JavaCompute node and in a Java user-defined node.
   - Define an ODBC connection if you want to interact with a database in a node that supports ESQL, including a JavaCompute node in which you use the `MbSQLStatement` interface.

3. Authorize the integration node to access the database.

   Access to a database from within a message flow is controlled by user ID and password.

   - Linux UNIX Use the **mqsisetdbparms** command to specify a user ID and password for a specific database, or to set up a default user ID and password.
   - Windows Use the **mqsisetdbparms** command to specify a user ID and password for a specific database, or to set up a default user ID and password. If Windows integrated authentication is being used for SQL Server database access, then the service user ID under which the broker process runs are used by Windows to access the SQL Server database. That is, it ignores any user ID and password credentials that were set using the **mqsisetdbparms** command.
   - To check existing accesses, use the **mqsireportdbparms** command.

Accessing databases from ESQL

Configure your integration server and your database to support connections from message flows.

Before you begin

- Run the **mqsicreateworkdir** command to create an integration server work directory.

About this task

You must configure your integration server and your databases to support read, write, and update operations in your message flows.

For details of the ESQL statements and functions that you can use to access databases, see "Interaction with databases using ESQL" on page 1666.

Procedure

- Set the `Data Source` property of each message flow node to the name (that is, the ODBC DSN) of the database that you want to access.

You can access more than one database by using the `FROM` clause in your ESQL statement, but all databases that are accessed from the same message flow node must have the same ODBC functions as the database that is specified on the `Data Source` property on that node. This requirement is always satisfied if the databases are of the same type (for example, DB2 or Oracle), at the same release level (for example, release 9.1), and on the same platform. Other database combinations might have the same ODBC functions. If a message flow node tries to access a database that does not have the same ODBC functions as the database specified on the `Data Source` property on that message flow node, the integration node generates an error message.

- Configure the integration server to be able to connect to the database:
  a) Create ODBC data source connections on the system on which the integration server is running.
  b) Define a user ID and password to be used by the integration server to connect to the database by using any of the following options:
     - To set a user ID and password for a particular database, use the **mqsisetdbparms** command.
     - To define default values for user ID and password for the integration server to use for all data source names for which you have not set specific values, use the **mqsisetdbparms** command to specify `dsn::DSN`.
     - Windows On Windows, if Windows integrated authentication is being used for SQL Server database access, then the service user ID under which the broker process runs is used by

Windows to access the SQL Server database. That is, it ignores any user ID and password credentials that were set using the **mqsisetdbparms** command

  - If you have not set up a default user ID and password:
    - On Windows, the service user ID and password are used to connect to the database.
    - On other platforms, connection to the database fails.
  - If you have set a specific user ID and password and want to check what the values are, use the **mqsireportdbparms** command.
- Set up the authorization for the user ID to access the database by using the administration facilities that are provided by the database vendor.

  If you do not do so, the integration server generates an error when the message flow runs.

What to do next

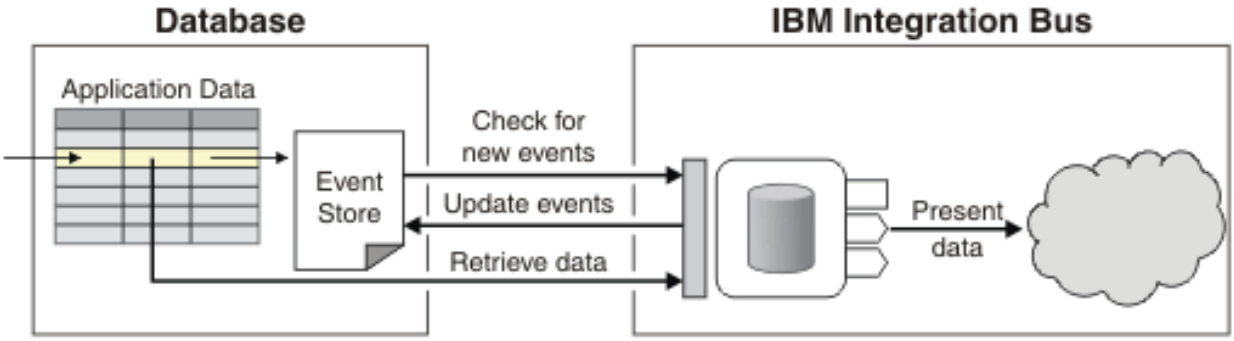
Note: With a single `SELECT FROM` clause, you can access only tables that exist in a single database.

If you access database columns that have names that are composed of only numeric characters, you must enclose the names in double quotation marks; for example, "0001". Because of this restriction, you cannot use a `SELECT *` statement, which returns the names without quotation marks; the names are therefore invalid and the integration server raises an exception.

Event-based database integration

Use a DatabaseInput node to respond to events in a database. For example, the integration node can keep an external system synchronized with a database by sending updates to the target system whenever data is changed in the database.

The database must record the fact that data has changed in an *event store*, which is typically a database table. The event store is not the same as the application data. The following diagram shows the interaction between the database, event store, and the integration node.



1. A database application changes a database table.
2. The database management system (DBMS) records the change in the event store.
3. The integration node polls the event store after the interval specified in the `Polling interval` configuration setting.
4. IBM App Connect Enterprise retrieves the new or changed data, and updates the event store, so that the data is processed only once.
5. IBM App Connect Enterprise processes the data, and ultimately presents it to the target application, for example SAP, web services, or CICS Transaction Server for z/OS. The data can be presented in a different logical and physical format, if required.

Implementing

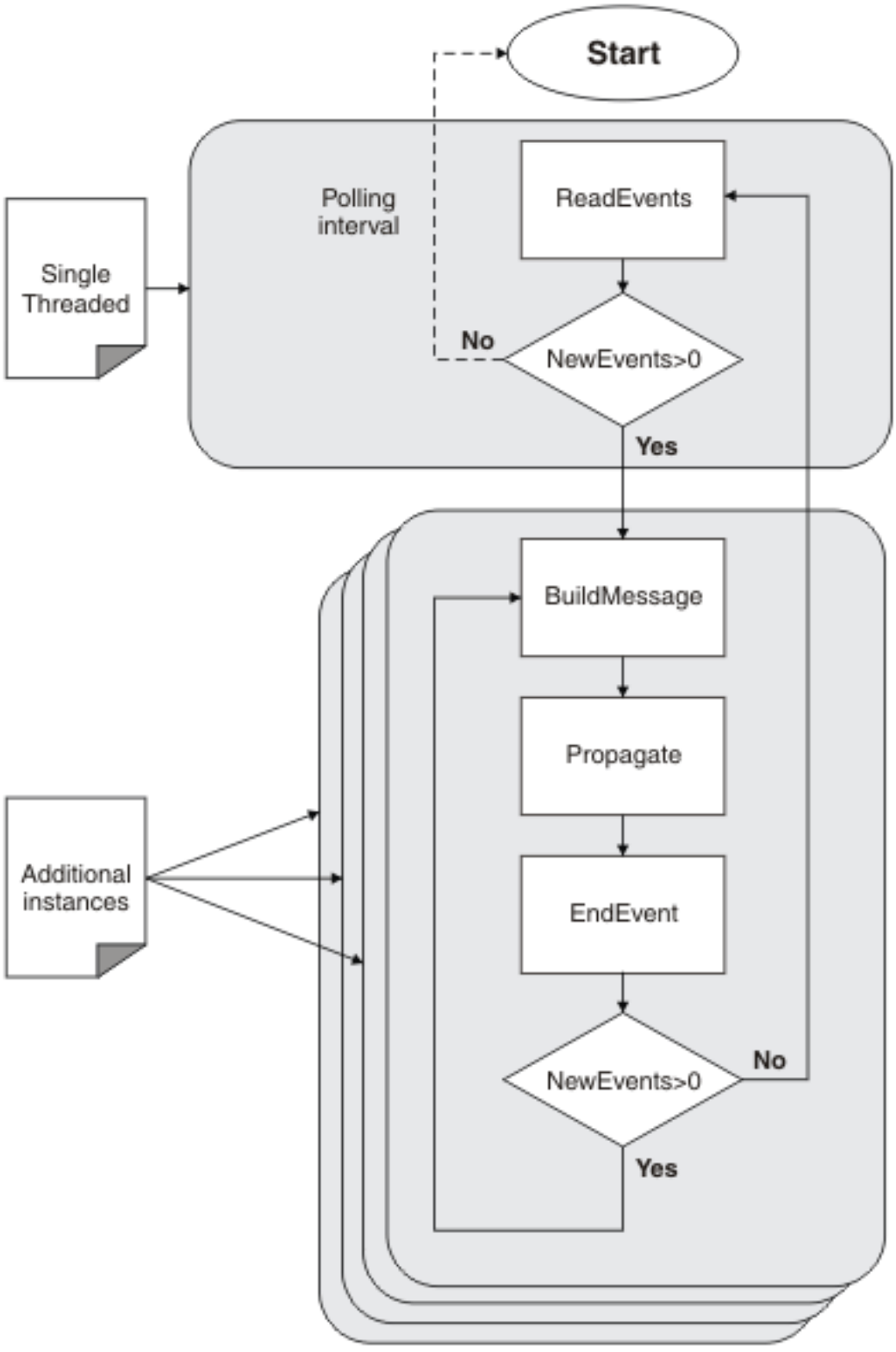
Implementing this scenario involves the following steps:

1. Configuring the database to record events.

2. Determining the format in which the target system must receive the data from these new events.
3. Configuring the integration node to detect these events by using the DatabaseInput node. To configure the DatabaseInput node, see “Configuring a DatabaseInput node” on page 1140.
4. Configuring the rest of the message flow to present the data to your target system in the correct format.

The DatabaseInput node

The following diagram shows how the DatabaseInput node works.



When the process starts, ReadEvents checks the event store for new events, which are then used by BuildMessage to build the message. This message is propagated to the message flow and then EndEvent updates the event store to ensure that the event cannot be processed again. When all events have been processed, the integration node calls ReadEvents to retrieve any events that have been added since the previous check. If the event store is empty, the integration node waits until the polling interval has expired, and then calls ReadEvents again. To avoid contention, the check of the event store is single-threaded.

For each event that is read by ReadEvents, BuildMessage builds the message that is propagated to the message flow. Building the message typically uses the event data to look up data in the application table. The data from the application table is then used to construct the message. When BuildMessage ends, the message is automatically propagated to the message flow. When the message is propagated, the integration node starts any downstream nodes that are required to process the message.

After the message has been propagated to the message flow, EndEvent updates the event store to ensure that the event that has just been processed cannot be processed again.

The detailed operation of ReadEvents, BuildMessage, and EndEvent is controlled by ESQL code. The DatabaseInput node contains an ESQL module with sample code and comments, which you must modify to suit your requirements. ReadEvents populates a NewEvents structure with event data that is read from the event table. Each event data row that is generated has a `Key` and a `Usr` field name; `NewEvents.Event[].Key` and `NewEvents.Event[].Usr`. The `Key` field contains a unique key for an event, and is used to prevent duplicate event processing. The `Usr` field is typically a subtree that contains user-defined data that is associated with an event. For information about modifying the ESQL, see “Configuring a DatabaseInput node” on page 1140.

Transactions and Scaling

The processes that are completed by the DatabaseInput node are split across separate transactions. A new transaction is started when ReadEvents starts. When ReadEvents ends, this transaction is committed and new events are marked for processing. By committing this transaction, any locks put on the database by ESQL code that is run from ReadEvents are released. Then, for each event received by BuildMessage, a new transaction is started. This new transaction is committed after EndEvent finishes.

To scale a DatabaseInput node for many events, change the `Additional instances` property on the **Instances** tab from its default value of `0` to the number of instances that you require. If you are using additional instances, the database must be configured so that multiple applications can read different rows from the application table at the same time. ReadEvents always runs in single-threaded mode to avoid database contention, even if you use additional instances. To improve performance, ReadEvents can read multiple events each time it runs, and these events can be processed at the same time by multiple instances of BuildMessage. The event store must have a primary key, which ReadEvents uses to identify events that are currently being processed. You do not have to write the ESQL in ReadEvents to filter out events that are currently being processed by the message flow.

Failures and retry mechanism

- If an error occurs during ReadEvents, an error is logged to the system log. It will retry after the polling interval next expires. No message is propagated to the Failure terminal.
- If an error occurs during BuildMessage processing, the exception is propagated to the Failure terminal if attached. If exceptions are not handled on the failure terminal (that is, it is not wired or an exception is thrown from the failure terminal), then the transaction is rolled back. Otherwise, the transaction is committed. In either case, whether the exception is handled or not, the message is not propagated to the Out terminal and EndEvent is not invoked.
- If an exception is thrown downstream from the Out terminal, any exceptions not handled by the Catch terminal will cause the transaction to be rolled back and EndEvent is not invoked. If exceptions are handled on the Catch terminal, then the EndEvent is invoked and the transaction is committed.

Retry processing allows the application developer to cope with transient errors that occur in downstream nodes that have not been successfully handled by catch processing. The input node can be configured with one of three retry mechanisms:

1. `Failure`
2. `Short retry then failure`
3. `Short then long retry`

If '`Failure`' is selected, the event is moved to the '`Failed`' state and the transaction is rolled back. Failed events are detected and dealt with before dispatching any Ready events. The failed event is propagated to the Failure terminal, if wired, with an exception list.

For 'Short retry then failure', the message flow performs a number of retries equal to the **Retry Threshold** property, propagating the message down the Out terminal with an interval between each retry equal to the 'Short retry interval'. If the retry threshold is met without successful completion of the transaction, the message is then propagated to the Failure terminal.

For 'Short then long retry', the retry logic is the same as 'short retry then failure' up to the point that the retry threshold has been met. Then, instead of ending the retry attempts and propagating to the failure terminal, it will continue to retry indefinitely but now with a delay between each retry equal to the 'Long retry interval'.

The EndEvent operation is not automatically invoked if the transaction has been rolled back, or if the message has been propagated to the failure terminal. In these circumstances it is up to the user to remove or update the event in the event store if they wish to prevent it from being re-processed.

The EndEvent operation is automatically invoked if the transaction completes successfully, including when an exception has occurred downstream of the Database Input node but that exception has been handled successfully down a Catch terminal.

Configuring a DatabaseInput node
Create and configure message flows that respond to events in a database.

Before you begin

Read the following topics:

- "Event-based database integration" on page 1136
- "Responding to database updates" on page 1142

Check which databases are supported on which platform, and if any restrictions apply:

- IBM App Connect Enterprise system requirements

Ensure that your database is configured to record events (uses an event table), and that you know how to query those events.

z/OS If you use DB2 on z/OS, your user ID (or your user group) requires permission to perform a SELECT on SYSIBM.SYSJAROBJECTS.

Complete the following tasks:

- Add a database definition to the IBM App Connect Enterprise Toolkit.

About this task

When you drag a DatabaseInput node onto the canvas, IBM App Connect Enterprise creates an ESQL module that contains boilerplate text. To configure the DatabaseInput node, modify the statements in that module to suit your requirements.

When you double-click the node to modify the ESQL code, the editor displays the **Database Event Design** tab for the module. Complete the mandatory fields and then click **Generate query**. To view or modify the code, click the **Source** tab. Code that has been generated is clearly marked by color-coded `--@!{` and `--@!}` comments. Any changes that you make within these comments are lost if you regenerate the code.

Procedure

1. In the IBM App Connect Enterprise Toolkit, drag a DatabaseInput node onto the canvas, and double-click the node.

   The **Database Event Design** tab is displayed. Ensure that the correct module is selected.

2. Complete the **Event Table** section.
   a) Optional: Complete the `Database schema` property.
      Leave it blank to use the default runtime schema.
   b) Complete the `Table` property.
      This property represents the database table used as your event store.
   c) Complete the `Primary key` property.
      This property represents the primary key of the database table used as the event store.
   d) Complete the `Foreign key to application table` property.
      This property represents the column in the event table that references the row in the application table containing the changed data to be processed by the DatabaseInput node. This is typically the primary key of the application table.
   e) Optional: Complete the `Status column` property.
      This property represents the name of a column, if you update a column in the event table to indicate that the event has been processed. Leave blank if you delete events from the event table after processing.
   f) Optional: Complete the `New event status value` property.
      This property represents the value written to the status column when the event is first added. Enclose character values in single quotation marks, for example 'Y'. Enter numbers without quotation marks. For a null value, enter NULL. Check the trigger setting in your database for appropriate values.
   g) Optional: Complete the `Processed event status value` property.
      This property represents the value written to the status column after the event has been processed. Enclose character values in single quotation marks, for example 'Y'. Enter numbers without quotation marks. For a null value, enter NULL. Check the trigger setting in your database for appropriate values.
3. Complete the **Application Table** section.
   a) Complete the `Table` property.
      This property represents the table that includes the changed data to be processed by the DatabaseInput node.
   b) Complete the `Primary key` property.
      This property represents the primary key of the database table used as the application table.
   c) Complete the `Output message element` property.
      This property represents the output message that will be propagated to the flow.
4. Click **Generate query**.
5. Optional: Click the **Source** tab to view the code, or add customized code.
6. On the **Basic** tab of the DatabaseInput node, specify the data source. This data source is the ODBC data source name of the database that contains the tables that you refer to in the ESQL module.
7. On the **Basic** tab, ensure that the `ESQL module` property refers to the correct module.
8. Optionally, change values on the other tabs of the node.
9. Configure the rest of the flow to use the message from this node.

What to do next

Configure your target system to receive the message.

Changing the default color of auto-generated text

Procedure

1. Click **Window** > **Preferences**.

2. In the tree on the left, navigate to **Integration Development** > **ESQL** > **ESQL Editor**.
3. On the **Colors** tab, select **Auto-generated**, and select the color.

Responding to database updates
Implement a message flow that responds to database updates, and presents the data to another application.

Before you begin

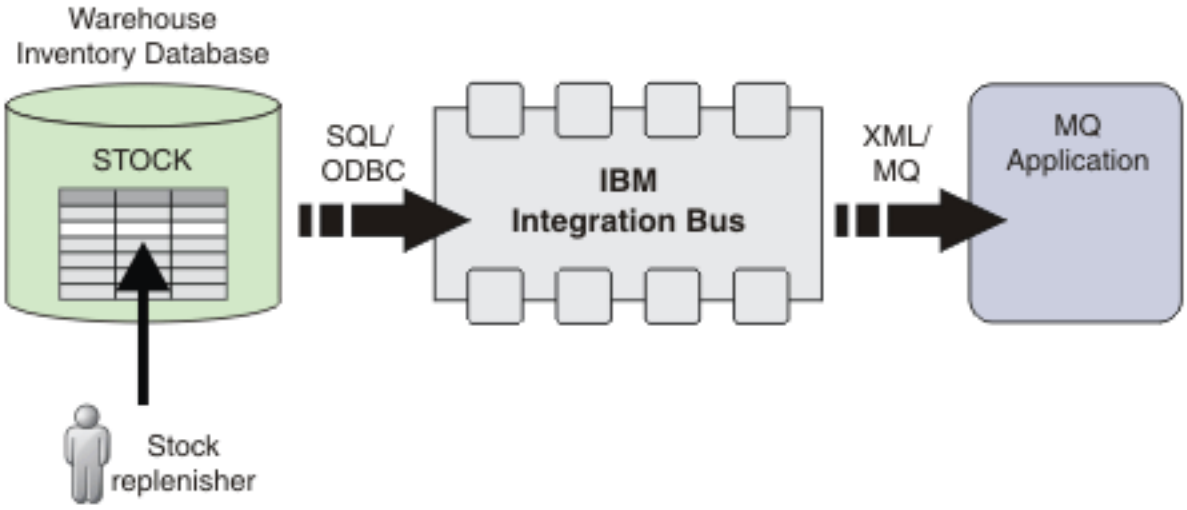
- Create an event table (a database table that serves as a transient store for event data).
- Create a trigger on the application data table. The trigger inserts a row into the event store whenever the application data is changed.
- Configure the IBM App Connect Enterprise runtime component to connect to the database.

z/OS If you use DB2 on z/OS, your user ID (or your user group) requires permission to perform a SELECT on SYSIBM.SYSJAROBJECTS.

You do not need experience of ESQL to complete this task.

About this task

Scenario: A retail company uses a relational database to manage its stock inventory. Since a recent acquisition, a new set of applications based on XML and IBM MQ are added to the environment. The applications notify interested parties of any changes to the stock levels. The applications have a predefined XSD schema model that describes the input message.



IBM App Connect Enterprise is used to respond to database updates, and to notify the IBM MQ application of these changes.

1. A DatabaseInput node retrieves the data.
2. A transformation node, such as a Compute node or a Mapping node, transforms the data to the target format.
3. An output or request node, such as an MQOutput node, sends the transformed message to the target system.

You will complete the following actions:

1. “Discover the database model” on page 1143
2. “Create a new message model for the database input” on page 1143
3. “Create the message flow” on page 1143
4. “Test the flow” on page 1144

Discover the database model
Create a `.dbm` file that you will use to create the message model. You create a data design project, and use a wizard to give IBM App Connect Enterprise details of your database event store and data table.

Procedure

1. Click **File** > **New** > **DataBase Definition**.
2. Click **New** to create a new data design project, or select an existing data design project from the drop down list.
3. Select the appropriate database type and version, and then click **Next**.
4. Select an existing JDBC connection, or create a connection to your database. If you create a connection, test the connection.
5. Select the database schema that you will use to create the message definition.
6. Select the database elements that you need for the model. You require `Tables` and `Triggers`.

   The data model is created, and you can see details of the database tables that are described in the chosen schema.

Create a new message model for the database input
Create a new message schema model file from the discovered data definition if you require a model of the data structure that the database input will present. You need a model if you want to graphically map this input. The model also enables content assistance auto-completion of paths in the ESQL editor if you are transforming the data in ESQL.

Procedure

1. Click **File** > **New** > **Message model**
2. In the **Other** section, select `Database record`, and then click **Next**.
3. Select `Create an XML schema file from a database definition`, and then click **Next**.
4. Navigate to and select the Database definition `.dbm` file that you created during discovery, and then click **Next**.
5. Ensure that the database tables that will be used are selected.
6. Click **Next** and then **Finish**.

Results

The New Message model wizard creates an XML schema message model file in your selected location.

Create the message flow
Create and configure a flow that consists of a DatabaseInput node, a Mapping node, and an MQOutput node. You will use the schema file that describes the input message to create a message definition file.

Procedure

1. Create an integration project that references both the data design project, and the message set project, that you created earlier.
2. Create a message flow, and drag a DatabaseInput node onto the canvas.
3. Configure the node as follows:
   a) Set the **`Data source`** to the ODBC connection that you created earlier.
   b) Follow the instructions in "Configuring a DatabaseInput node" on page 1140 to configure the ESQL procedures to provide details of the event store and application data.
4. To enable graphical mapping from the database data to the output message format, in addition to Creating the new message model for the database input as above, you also require a message model for the target message. This can be a DFDL or XML schema message model as appropriate to your scenario.

5. Drag a Mapping node onto the flow, and configure it.
   a) Set the map source to the message that you defined in "Create a new message model for the database input" on page 1143.
   b) Set the map target to the TARGET message that you just defined.

   You can use other methods to transform the message; see "Transforming and enriching messages" on page 1249.
6. Drag an MQOutput node onto the canvas, and set the queue manager name and queue name.

Test the flow
Use the debugger to test the flow.

Procedure

1. Start the debugger, and then add breakpoints to the flow.
2. Deploy the flow.
3. Change the data source, for example by adding a new row.

   You can change data from within the IBM App Connect Enterprise Toolkit. Under **Data Source Explorer**, right-click the table and choose **Data** > **Edit**.
4. Use the debugger to check that the flow is working correctly.

Event tables
An event table stores information about changes to application tables.

The event table is a database table created by the user, generally within the same schema as the application table for which it stores events. The event table describes the type of change made to an application table, and also contains an identifier for the changed row.

To populate an event table, one or more *triggers* must be created. A trigger is a database construct that can run an SQL script when a predefined action occurs. For example, a trigger can insert a row in the event table when an update in the application table occurs.

The following table shows some typical columns in an event table, and the reasons for including them.

Column name	Column function	Example value
EVENT_ID	Required. The primary key, which identifies the event being processed.	`1`
OBJECT_KEY	Required. The identifying element of the changed row in the application table, typically the element of the row in the primary key column.	`cust1`
OBJECT_VERB	Optional. The change performed, typically one of CREATE, UPDATE, or DELETE. This event is used to distinguish a DELETE event, where the application table contains no row to retrieve when the message for the flow is built.	`CREATE`
OBJECT_NAME	Optional. The name of the application table that has changed. This column is required if the DatabaseInput node is being used to support updates to more than one application table.	`customer`
EVENT_PRIORITY	Optional. The priority of the event. For example, you can ensure that high value transactions are computed first.	`1`
EVENT_TIME	Optional. The time at which the operation was performed. Generally used for logging or performance monitoring of the flow.	`2010-10-19T17:10:00`

Column name	Column function	Example value
EVENT_STATUS	Optional. Used to determine if the event has already been processed. Required if the events are not to be deleted or archived after processing.	`0`
EVENT_COMMENT	Optional. Free-form field, for example, it can be used to store the outcome of the message processing if the event was not deleted after processing.	`Processed with exceptions`

The column names are examples only. You can use other names. If you have a high-throughput application table, a single row might be changed multiple times between retrieving events from the event table. In this case, only the details of the latest change are picked up by the flow. If a record of intermediate changes is required, include more details in the event table. Also ensure that your event table has enough information about events generated by DELETE operations. Here, because the row in the application table no longer exists, all information required to successfully process the event must be present in the event table.

For example, if a new customer with primary key `cust1` is created in the application table, a row is added to the event table:

EVENT_ID	OBJECT_KEY	OBJECT_VERB
1	cust1	Create

The DatabaseInput node responds to the change, and processes the new row in a message flow.

Processing options on completion

When the message flow has processed an event, the flow can handle the event in the following ways:

- Delete the event. Use this option if you do not want to store the event for future reference.
- Update the status column. Use this option if you want to keep a record of processed events. Your event table must have a status column.
- Archive the event to a separate event table. Use this option if you want to keep a record of events while keeping the event table to a minimal size.

Working with IMS

You can use the IMSRequest node to connect to IMS, a message-based transaction manager and hierarchical-database manager for z/OS.

About this task

This section contains the following concept information:

- "IBM Information Management System (IMS)" on page 1146
- "IMS nodes" on page 1147
- "IMS transactions and programs" on page 1148
- "Response models" on page 1149
- "IMS message structure" on page 1150
- "IMS connections" on page 1151

This section contains the following tasks:

- "Preparing the environment for IMS nodes" on page 215
- "Propagating security credentials to IMS" on page 1152

IBM Information Management System (IMS)

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

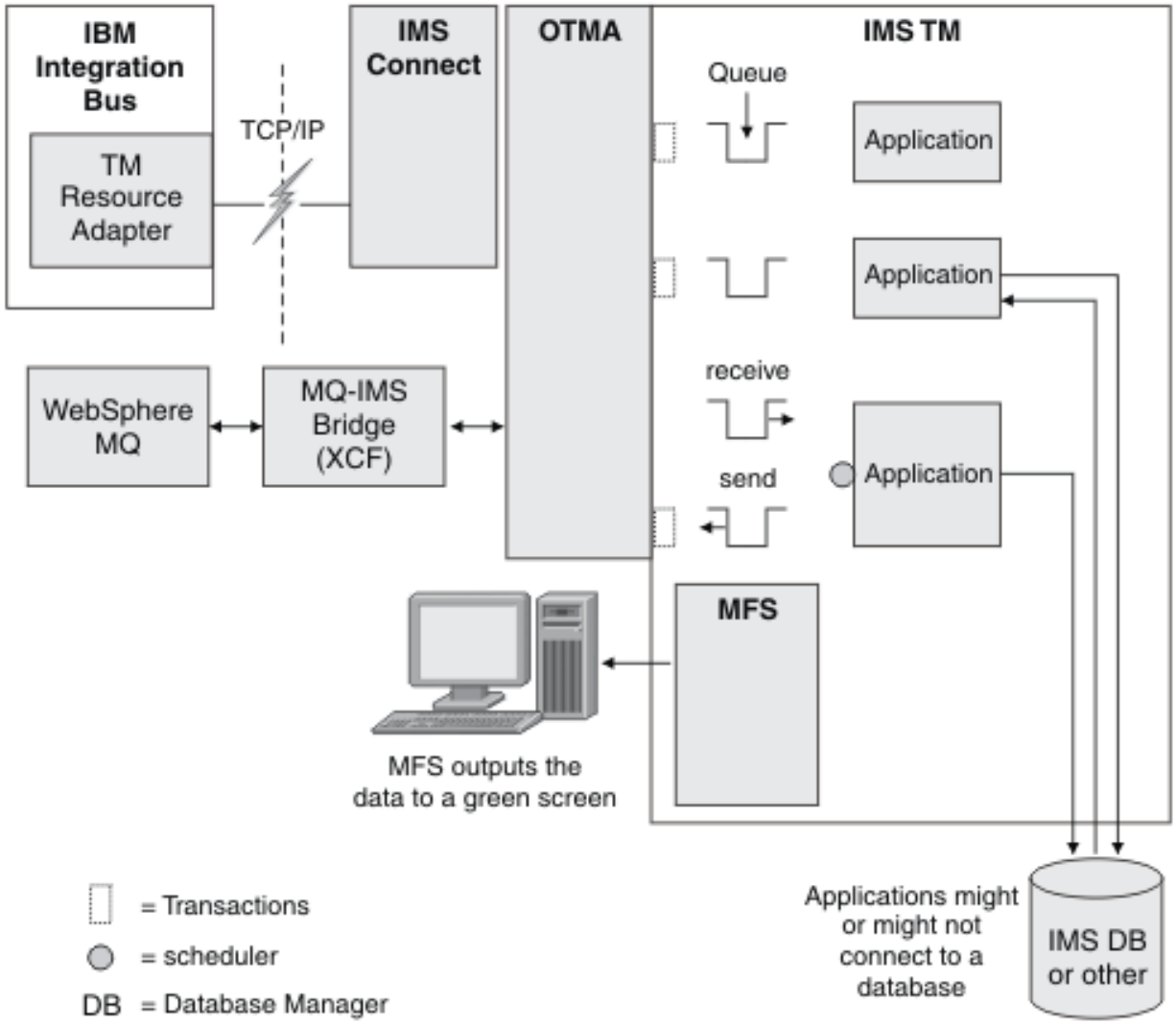
IMS includes two components:

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

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

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

The following diagram shows the layers of communication between IBM App Connect Enterprise and IMS.



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

- **The IBM MQ-IMS bridge**

  The WebSphere MQ-IMS bridge is a component of IBM MQ for z/OS. You can use it to access applications on your IMS system from IBM MQ applications. For more information about the IBM MQ-IMS bridge, see IBM MQ product documentation online.

- **The IMS SOAP Gateway**

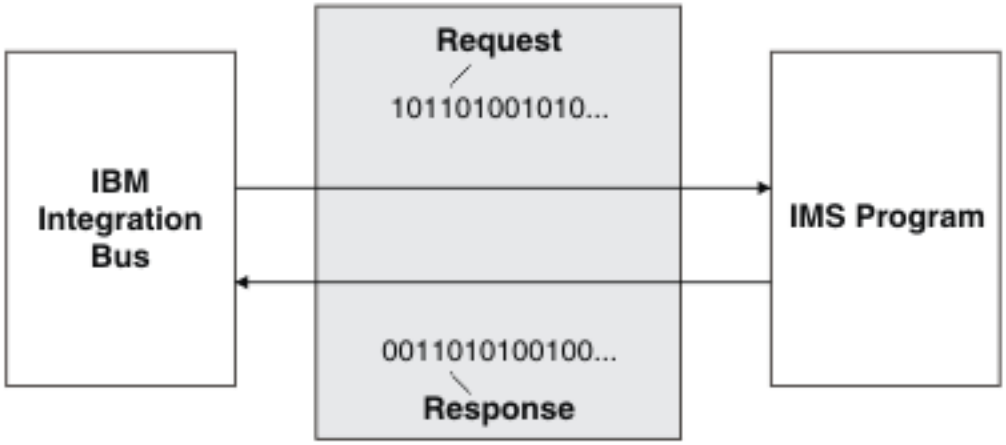
  The IMS SOAP Gateway is a web service solution that integrates IMS assets in a Service-Oriented Architecture (SOA) environment. For more information, see IMS SOAP Gateway web page.

For more details about how IMS works with IBM App Connect Enterprise, see “IMS nodes” on page 1147.

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

IMS nodes
IBM App Connect Enterprise message flows use IMS nodes to call programs that are running in IMS.

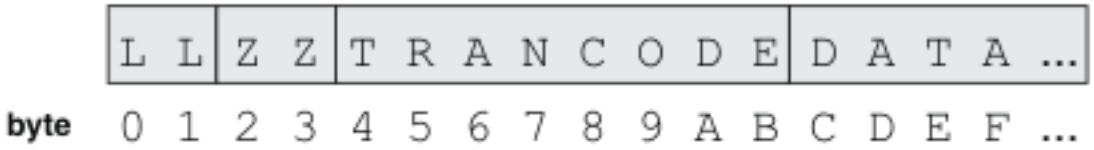
The IMS node sends a bit stream to IMS, which schedules one of its programs to process the message. The program generates a message, which IMS sends back to the IMS node, as illustrated in the following diagram.



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

Request bit stream

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

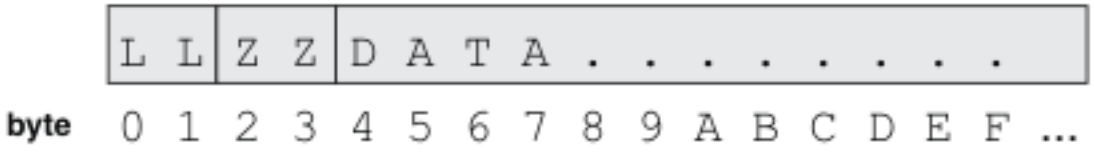


- *LLZZ* is a four-byte field. The first two bytes indicate the length of the bit stream, and the other two bytes are reserved for use by IMS.
- The transaction code can contain up to eight characters. If the code contains less than eight characters, the transaction code must be delimited by a space.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

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

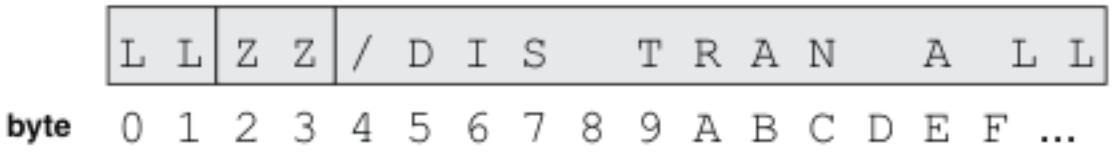
Response bit stream

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



Commands

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

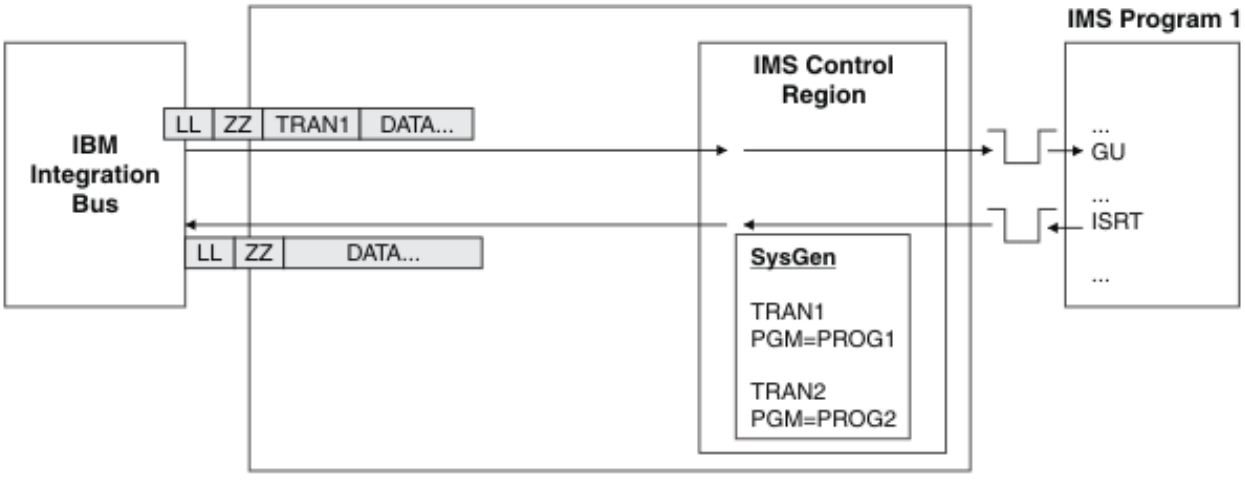


The first character after LLZZ is the slash (/) character, which is followed by the command verb and any arguments. For commands, the response bit stream has the same format as the response bit stream for transactions: LLZZ is followed by the response data.

For more information about IMS concepts, see the following topics:

- "IMS transactions and programs" on page 1148
- "Response models" on page 1149
- "IMS message structure" on page 1150
- "IMS connections" on page 1151

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



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

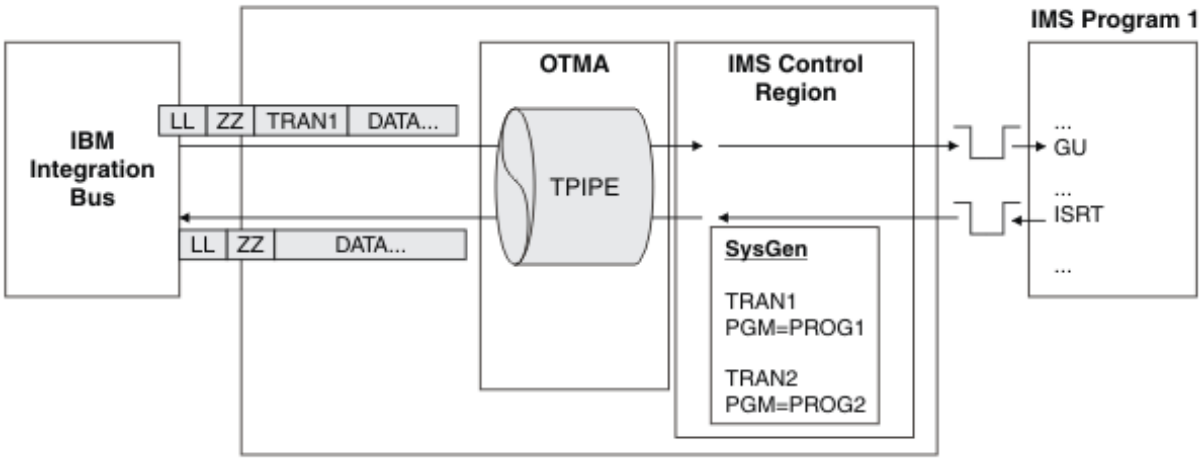
The IMS program gets messages by issuing a GU (GetUnique) call and it produces messages by issuing an ISRT (Insert) call. These calls are known as *DL/1 calls*. DL/1 is the programming interface to IMS. Other common DL/1 calls are PURG (purge) and GN (GetNext).

Response models
The synchronous request and response model is associated with IMS.

Synchronous request and response

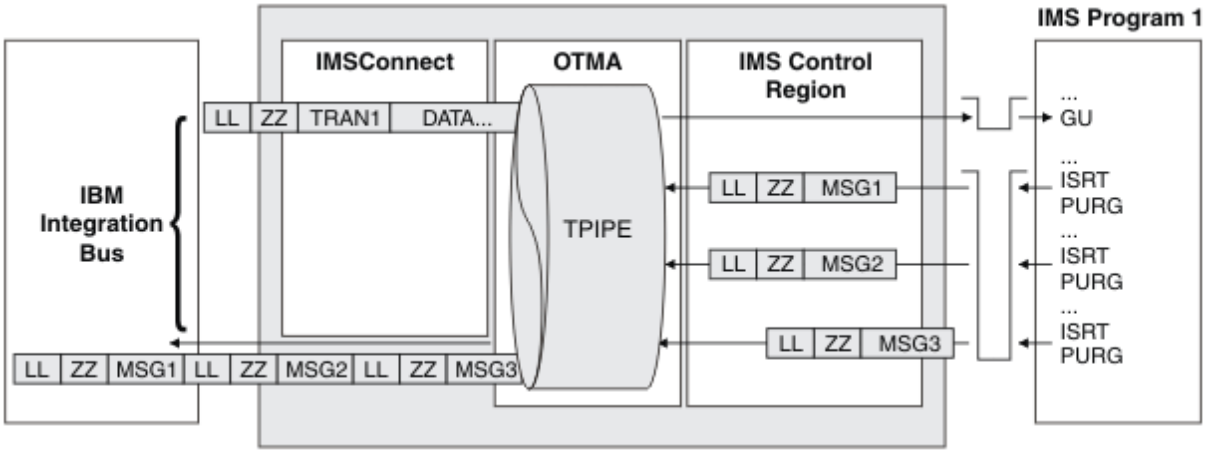
In the synchronous request and response model, the entire transmission is returned to the IMS node synchronously.

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



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

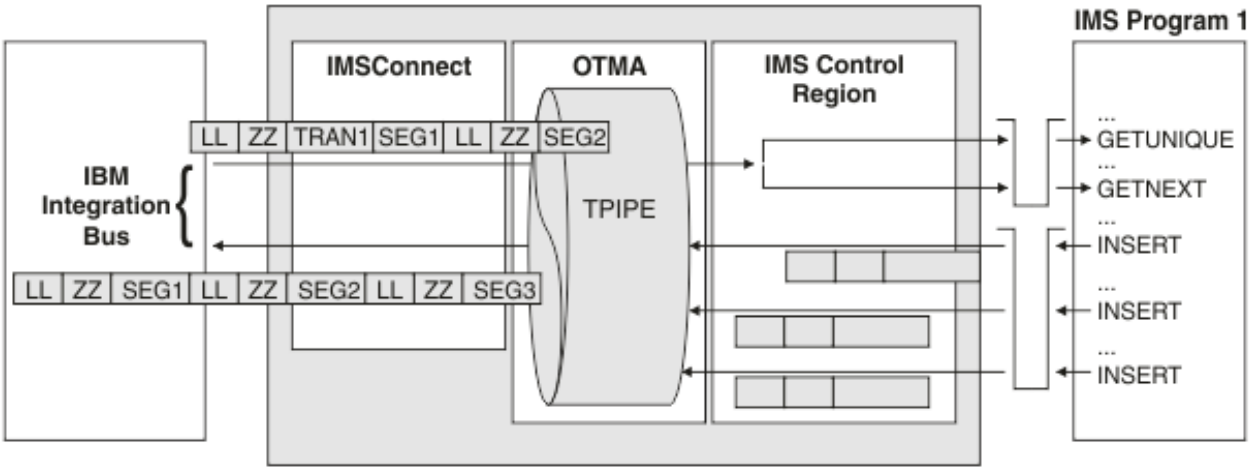
The output of a transaction can contain multiple messages. The IMS program inserts then purges each message, as shown in the following diagram. (For multi-segment output, the IMS program inserts multiple messages without purging them.) If the IMS program inserts multiple messages in a single sync point,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



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

The bit stream that flows between IBM App Connect Enterprise and the IMS program (also known as the *transmission*) can contain multiple segments. Each segment begins with the LLZZ and Transaction code fields that are described in "IMS nodes" on page 1147. The transmission can contain multiple messages, each one containing multiple segments. The IMS program gets the segments one at a time and typically inserts the output data onto the queue one segment at a time. The IMS program purges the end of a message before it sends the first segment of the next message.

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



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

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

An IMS transaction's response can have various structures:

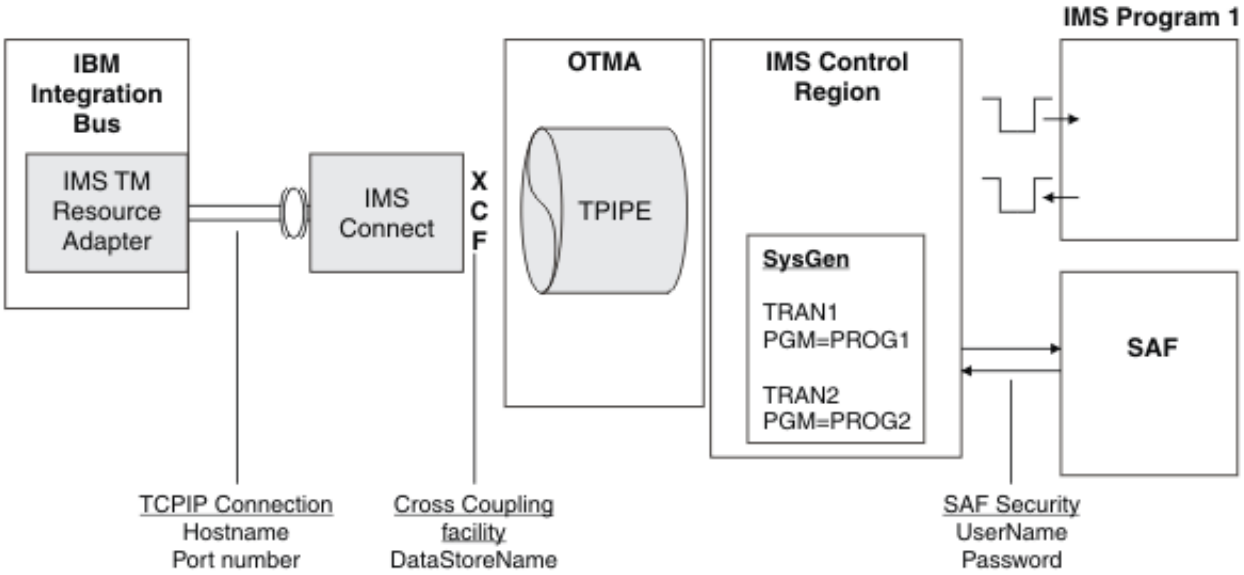
- An MRM-CWF structure, such as a message definition that is derived from a COBOL copybook
- An MRM-TDS structure when the output is 3270-based, such as a list of name=value strings
- Another structure, such as XML output from a Java program that is running in an IMS Java Processing Region (JPR)

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

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

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

IMS connections
Open Transaction Manager Access (OTMA) is used to provide access to IMS from IBM App Connect Enterprise.



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

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

The IMSRequest node can use an identity that is present on an input message, and propagate it to IMS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see "Propagating security credentials to IMS" on page 1152.

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

Preparing the environment for IMS nodes

Before you can use the IMS nodes, you must set up the runtime environment so that you can access the IMS system.

Before you begin

Read "IBM Information Management System (IMS)" on page 1146.

About this task

Complete the following steps to ensure that IBM App Connect Enterprise can connect to the IMS system.

Procedure

1. Ensure that IMS Connect is installed and started on the IMS system.
2. Use the **mqsisetdbparms** command to set security details in the integration server store.

   For example, to associate a user ID and password pair with an IMS Connect connection, run the **mqsisetdbparms** command as shown:

```
mqsisetdbparms -w c:\workdir\ACEServ1 -n ims::mySecurityIdentity -u myuserid -p mypassword
```

Propagating security credentials to IMS

The IMSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined for the node.

About this task

If an IMSRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to IMS.

Procedure

To propagate an identity to be used for the IMS request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the IMSRequest node, or create a security profile, by following the instructions in “Creating a security profile” on page 2584.
2. Use the BAR editor to select a security profile for the IMSRequest node that has identity propagation enabled.

   For detailed instructions, see “Configuring a message flow for identity propagation” on page 2613.

Working with CORBA

Use CORBA nodes to connect to CORBA Internet Inter-Orb Protocol (IIOP) applications.

About this task

This section contains the following concept information:

This section contains the following tasks:

Common Object Request Broker Architecture (CORBA)

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA is not associated with a particular programming language, and any language with a CORBA binding can be used to call and implement CORBA objects. Objects are described in a syntax called Interface Definition Language (IDL).

CORBA includes four components:

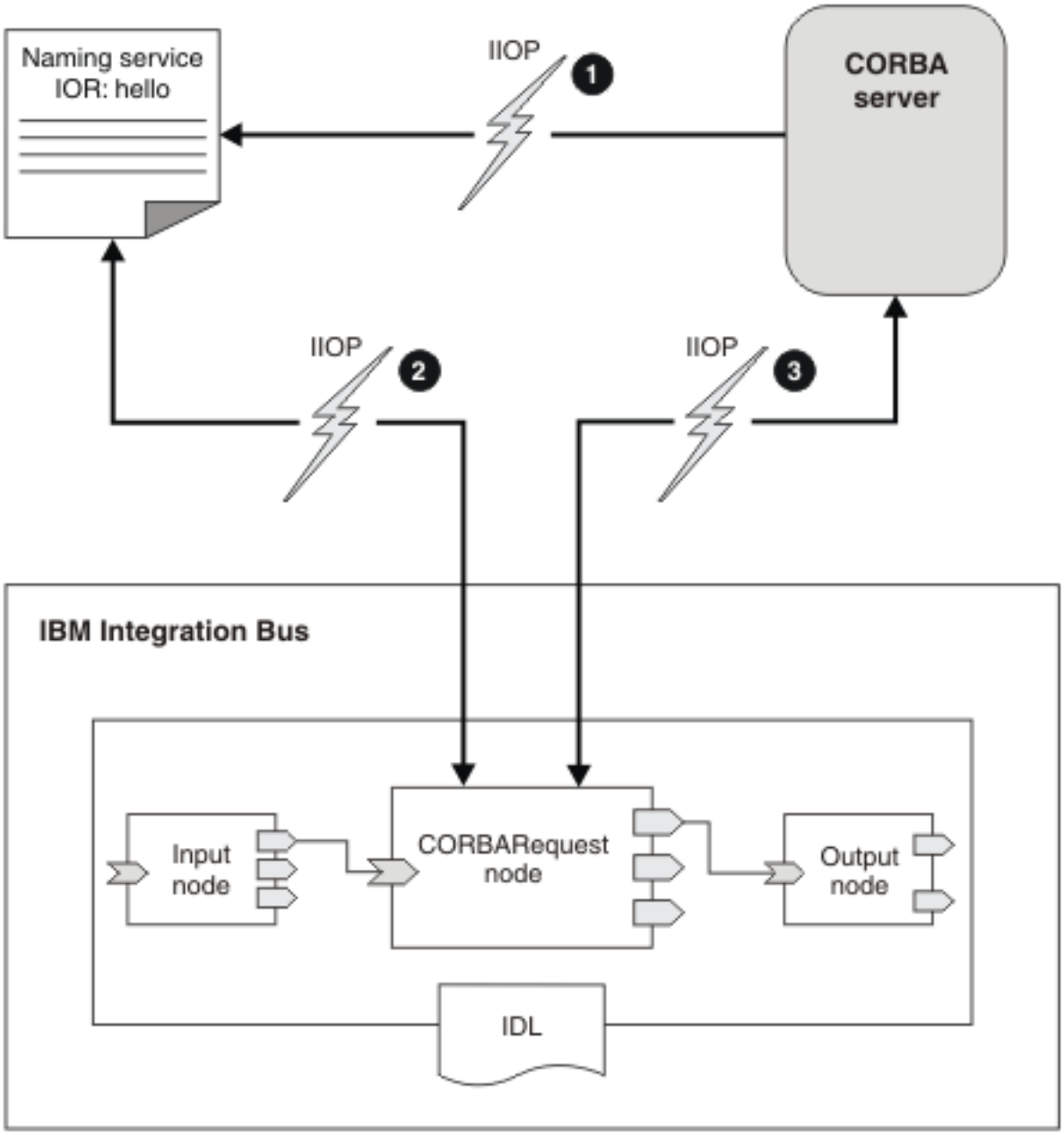
Object Request Integration node (ORB)
: The Object Request Integration node (ORB) handles the communication, marshaling, and unmarshaling of parameters so that the parameter handling is transparent for a CORBA server and client applications.

CORBA server
: The CORBA server creates CORBA objects and initializes them with an ORB. The server places references to the CORBA objects inside a naming service so that clients can access them.

Naming service
: The naming service holds references to CORBA objects.

CORBARequest node
: The CORBARequest node acts as a CORBA client.

The following diagram shows the layers of communication between IBM App Connect Enterprise and CORBA.



The diagram illustrates the following steps.

1. CORBA server applications create CORBA objects and put object references in a naming service so that clients can call them.
2. At deployment time, the node contacts a naming service to get an object reference.
3. When a message arrives, the node uses the object reference to call an operation on an object in the CORBA server.

For more details about how CORBA works with IBM App Connect Enterprise, see “CORBA nodes” on page 1154.

CORBA nodes

Use CORBA nodes to connect IBM App Connect Enterprise with CORBA Internet Inter-Orb Protocol (IIOP) applications.

CORBA is a standard for distributing objects across networks so that operations on those objects can be called remotely. CORBA objects are described in Interface Definition Language (IDL) files, and these IDL

files are used to configure the CORBA message flow nodes. The IDL file is stored in a message set project, in a folder called CORBA IDLs.

An IDL importer imports the IDL file into the message set project and creates the message definition file (`.mxsd`) in the message set. This message definition file is used for mid-flow validation, ESQL content assist, and the Mapping node.

For each IDL file, a single message definition is created. In the message definition, two messages are created for each operation in the IDL file: one message for the request, and one for the response. The request has a child element for each in and inout parameter; the response has a child element for each inout and out parameter, and a child element named "_return" for the return type of the operation.

The name of these elements is based on the interface name and operation name; for example, for the operation *sayHello* in the Interface *Hello*, the request element is called *Hello.sayHello*, and the response element is called *Hello.sayHelloResponse*. If the interface is contained in a module, the request and response element names are qualified with the names of the modules. For example, if the operation *sayHello* in the Interface *Hello* is contained in *ModuleB*, which in turn is contained in ModuleA, the response element would be called *ModuleA.ModuleB.Hello.sayHelloResponse*.

When you add a message flow that contains CORBA nodes to a BAR file, all the IDL files that are used by the nodes are added to the BAR file automatically.

The main scenario for connecting IBM App Connect Enterprise with CORBA applications is described in the following section.

IBM App Connect Enterprise calls a CORBA server

By using a message flow that includes a CORBARequest node, you can give existing CORBA applications a new external interface; for example, a SOAP interface. The message flow uses the IDL file to configure which operation is called on which interface.

After you have deployed the BAR file, you can start the message flow. A CORBA request is sent by using values from the message tree in the DataObject domain as input parameters. If a response is received, the return type and output parameters are propagated to the Out terminal of the CORBARequest node. You can use the data that is returned from the CORBA application to verify the result from the CORBARequest node.

CORBA support
The CORBA nodes in IBM App Connect Enterprise support a set of types and operations in imported IDL files.

IBM App Connect Enterprise supports the CORBA 2.3.1 specification and uses the Java 7 JRE ORB, and is therefore compatible with any CORBA vendor that is compatible with the JRE ORB.

IBM App Connect Enterprise currently supports the following CORBA types and operations.

- All primitive types, except bounded strings
- Two-way operations with in, inout, and out parameters
- User-defined exceptions
- Enums
- Modules
- Sequences (sequences must have associated typedefs; anonymous sequences are not supported)
- Structs
- Typedefs
- Comments

- The following preprocessing tokens:
  - #ifndef
  - #endif
  - #define
  - #include

  Other preprocessing tokens are ignored.

When you import an IDL file, supported and unsupported operations are listed. You can import and deploy IDL files that contain unsupported types and operations, but if you try to call one of these unsupported operations, you see an error message.

You cannot import an IDL file that is not valid. During the importing process, if you select an IDL file that is not valid, you see an error message and cannot complete the wizard.

Abstract interfaces and interfaces that contain inheritance are not supported. IDL pragma directives are not supported, but you can include the following pragmas in your IDL file (these pragmas are ignored):

- cpponly
- ID
- init
- localonly
- localonly abstract
- Prefix
- version

IDL data types
When you use the DataObject domain with CORBA, you need to know how XML schema and ESQL types correspond to the types in the IDL file.

Primitive IDL types

The following table shows the mapping between IDL types, XML schema simple types, and ESQL types.

IDL	XML schema	ESQL
boolean	xsd:boolean	BOOLEAN
char	<xsd:simpleType name="char"> <xsd:restriction base="xsd:string"> <xsd:length value="1" fixed="true/> </xsd:restriction> </simpleType>	CHARACTER
wchar	<xsd:simpleType name= "wchar"> <xsd:restriction base="xsd:string"/> </xsd:simpleType>	CHARACTER
double	xsd:double	FLOAT
float	xsd:float	FLOAT
octet	xsd:unsignedByte	INTEGER
long	xsd:int	INTEGER
Long long	xsd:long	INTEGER
short	xsd:short	INTEGER

IDL	XML schema	ESQL
string	xsd:string	CHARACTER
wstring	xsd:string	CHARACTER
Unsigned short	xsd:unsignedShort	INTEGER
Unsigned long	xsd:unsignedInt	INTEGER
Unsigned long long	xsd:unsignedLong	DECIMAL

Complex IDL types

IBM App Connect Enterprise supports the following complex IDL types:

- Enums
- Typedefs
- Sequences
- Structures

Each complex type is supported in the following places:

- Return types for operations
- In parameters
- Inout parameters
- Out parameters
- Inside exceptions
- Inside structures
- Inside sequences
- Inside typedefs

The following examples show the mapping between IDL types, XML schema, and XML.

Enums

IDL Enums are mapped to enumerations in XML schema. Enums inside the tree are of type string.

Here is an example IDL file:

```
enum myEnum {A, B, C};
interface example {
 void myoperation(in myEnum input1);
};
```

Here is an example XML schema:

```
<xsd:simpleType name="myEnum">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="A"/>
  <xsd:enumeration value="B"/>
  <xsd:enumeration value="C"/>
 </xsd:restriction>
</xsd:simpleType>
```

Here is an XML example:

```
<example.myoperation>
 <input1>A</input1>
</example.myoperation>
```

Sequences and typedefs

IDL typedefs are mapped to XML schema type restrictions. IDL sequences are mapped to XML schema sequence complex types. Sequences can be used only within typedefs.

Here is an example IDL file:

```
Typedef long myLong;
typedef sequence<long> longSeq;
interface example {
 void myoperation(in longSeq input1, inout myLong input2);
};
```

Here is an example XML schema:

```
<xsd:complexType name="longSeq">
 <xsd:sequence>
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded" type="xsd:int"/>
 </xsd:sequence>
</xsd.complexType>
```

A sequence can be bounded with the syntax sequence<long, 10>, which puts a bound in the XSD file.

Here is an XML example:

```
<example.myoperation>
 <input1>
  <item>10</item>
  <item>11</item>
  <item>12</item>
 </input1>
</example.myoperation>
```

Structures

IDL structures are mapped to XML schema complexType definitions.

Here is an example IDL file:

```
struct myStruct {
 char c;
 string str;
 octet o;
 short s;
 unsigned long long ull;
 float f;
 double d;
};
interface example {
 void myoperation(in myStruct input1);
};
```

Here is an example XML schema:

```
<xsd:complexType name="myStruct">
 <xsd:sequence>
  <xsd:element name="c" type="xsd:string" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="str" type="xsd:string" nillable="true" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="o" type="xsd:byte" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="s" type="xsd:short" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="ull" type="xsd:unsignedLong" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="f" type="xsd:float" maxOccurs="1" minOccurs="1"/>
  <xsd:element name="d" type="xsd:double" maxOccurs="1" minOccurs="1"/>
 </xsd:sequence>
</xsd:complexType>
```

Here is an XML example:

```
<example.myoperation>
 <input1>
  <c>c</c>
  <str>hello</str>
  <o>12</o>
  <s>10</s>
  <ull>110</ull>
  <f>12.0</f>
  <d>12.1</d>
 </input1>
</example.myoperation>
```

Modules

In CORBA, modules provide scope. If an interface is contained in a module in the IDL file, the interface name is qualified with the module name in the following format:

ModuleName.*InterfaceName*.*OperationName*

The following example shows a module in an IDL file.

```
Module one {
 Interface OneAInterface {
 };
};
```

The fully qualified name of the interface called OneAInterface is `one.OneAInterface`. In an IDL file, modules can be nested in other modules. In this case, the fully qualified name of the interface can include more than one module name, starting from the root module; for example:

ModuleNameA.*ModuleNameB*.*InterfaceName*.*OperationName*

An IDL file can contain more than one operation with the same name provided that the operations are in different modules.

CORBA naming service
A CORBA naming service holds CORBA object references.

A CORBA server puts references to CORBA objects inside a naming service so that clients can query the naming service and obtain the object reference, then call operations on the CORBA objects. Typically, a client queries the naming service once, then caches the object reference.

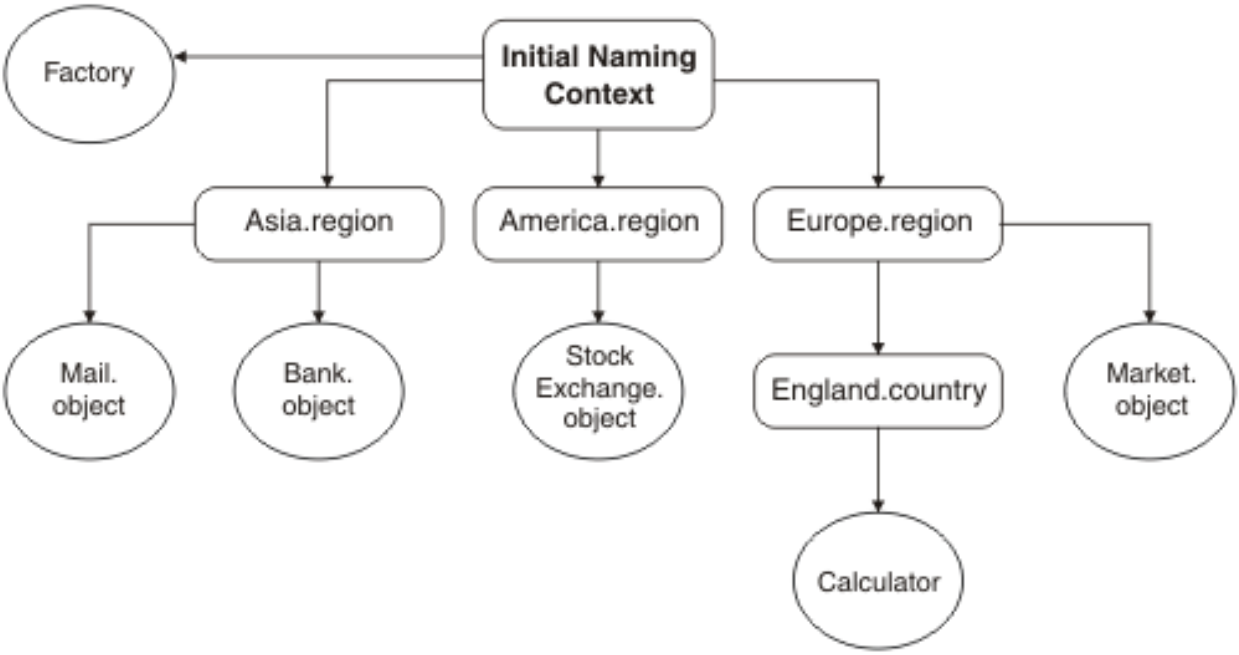
A CORBARequest node is a CORBA client; therefore, when it is deployed, the node contacts a naming service to obtain an object reference. If the object reference is not in the naming service at deployment time, or the naming service that is configured on the node is unavailable, the CORBARequest node issues a warning, and attempts to contact the naming service to get the object reference when it receives a message. If an object reference cannot be acquired from the naming service when the node receives a message, an error is issued. You can specify the location of an object reference by using the properties on the CORBARequest node. For more information, see node.

Identifying an object reference in a naming service

Each object in a naming service has a unique name. You must use this name when you configure the `Object reference name` property on the CORBARequest node.

Naming services are typically arranged in a hierarchy so that names can be given context or scope. The initial naming context is at the top of the hierarchy. Object references can be added to the initial naming context, and additional contexts can exist below it. The number of levels in the hierarchy is unlimited.

Object references and contexts can be assigned a *kind* to facilitate grouping. The kind is appended to the context in the format *context.kind*. If you are using IBM App Connect Enterprise to access an external CORBA application, you need to know the location of the naming service and the name of the object reference in the naming service. The following example shows how to determine the exact string representation of the name.



In the diagram, contexts are represented by squares, and object references are represented by circles.

- An object called Factory is directly attached to the initial naming context.
- Three contexts, with kind *region*, are also attached to the initial naming context.
- These three contexts each have one or more object references attached to them.
- The Europe context has an England context attached to it, of kind *country*, which has an object attached to it (*Calculator*).

The name that you specify when you configure the `Object reference name` property on the CORBARequest node reflects the position of the object in the hierarchy. The following table shows how to refer to the specific objects in the diagram.

Object	**Object reference name**
Factory	Factory
Bank	Asia.region/Bank.object
Mail	Asia.region/Mail.object
StockExchange	America.region/StrockExchange.object
Market	Europe.region/Market.object
Calculator	Europe.region/England.country/Calculator

All objects in the naming service can be connected directly to the initial naming context; in which case, their names would be in the same format as the Factory object in this example.

CORBA operation parameters
CORBA operations can have parameters that can be modified by the server.

CORBA operations can have in, out, and inout parameters. In and inout parameters dictate the appearance of the tree under the DataObject domain when going into a CORBARequest node. The return type, inout and out parameters dictate the appearance of the tree when leaving a CORBARequest node.

The inbound message is contained in the element *interfaceName.operationName* and needs an element for every in or inout parameter. If the interface is contained in a module, the name is qualified with the name of the module. If the module is nested in other modules, all module names are stated; for example: *moduleNameA.moduleNameB.interfaceName.operationName*. Out parameters are not required because

the client does not send a value for out parameters. These elements must be in the same order as the parameters in the IDL file.

The output message from the CORBARequest node is contained in the element *interfaceName.operationName*Response. If the interface is contained in a module, the name is qualified with the module name. The outbound message has an element for the return type, named *_return*, and an element for every inout and out parameter.

Here is an example of an IDL file:

```
interface exampleInterface {
 string outsideModuleOperation(in string one, out string two, inout string three);

};
```

The input message might look like the following example:

```
<exampleInterface.outsideModuleOperation>
 <one>something</one>
 <three>something</three>
</exampleInterface.outsideModuleOperation>
```

The output message might look like the following example:

```
<exampleInterface.outsideModuleOperationResponse>
 <_return>something</_return>
 <two>something</two>
 <three>something</three>
</exampleInterface.outsideModuleOperationResponse>
```

The input message requires all in and inout parameters, therefore one and three are specified. The output has the following elements:

- A *_return* element (because the operation has a return type of string)
- An element called *two* (because *two* is an out parameter)
- An element called *three* (because *three* is an inout parameter)

User-defined exceptions

Exceptions are propagated to the Error terminal under the DataObject domain; the structure of the message depends on the exception.

The following example shows how a user-defined exception is defined in an IDL file.

```
exception BadRecord {
    string why;
  };


  interface SomeInterface {
    long bar(in float pi) raises (BadRecord);
  };
```

The operation *bar* can issue the exception *BadRecord*. If this exception is issued, the following message is propagated to the Error terminal.

```
<BadRecord>
 <why>Reason text</why>
</BadRecord>
```

Edge cases

Two identified edge cases exist for reading a tree and producing a tree:

- No input parameters exist.
- A void function has no inout or out parameters

The following sample IDL file illustrates two examples.

```
interface exampleInterface {
 string exampleOne();
void exampleTwo(in string one);
};
```

- **Example 1 - no input parameters exist**

  Input message: The input message is irrelevant because the CORBARequest node does not look at the body of the message.

  Output message:

  ```
  <exampleInterface.exampleOneResponse>
   <_return>something</_return>
  </exampleInterface.exampleOneResponse>
  ```

- **Example 2 - A void function has no inout or out parameters**

  Input message:

  ```
  <exampleInterface.exampleTwo>
   <one>something</one>
  </exampleInterface.exampleTwo>
  ```

  Output message:

  ```
  <exampleInterface.exampleTwoResponse>
  </exampleInterface.exampleTwoResponse>
  ```

Connecting to an external CORBA application

Connecting to an external CORBA application involves importing an IDL file, creating a message flow, building a message, and processing the response from the CORBARequest node.

Before you begin

An IDL file is used to configure the CORBARequest node. Ensure that you have a valid IDL file that contains elements that are supported by IBM App Connect Enterprise. The CORBA IDL file must contain at least one interface that has one operation. For more information, see “CORBA support” on page 1155.

About this task

To connect to an external CORBA application, complete the following steps.

Procedure

1. Create an integration project.
2. Import an IDL file into the IBM App Connect Enterprise Toolkit, as described in “Message Sets: Importing an IDL file” on page 2309.
3. Develop a message flow that contains a CORBARequest node, as described in “Developing a message flow with a CORBARequest node” on page 1163.
4. Build a message to send to the CORBARequest node, as described in “Building a message for the CORBARequest node” on page 1164.
5. Process the response from the CORBARequest node, as described in “Processing responses from a CORBARequest node” on page 1167.

Developing a message flow with a CORBARequest node
To connect to an external CORBA application, create a message flow that contains a CORBARequest node.

Before you begin

Ensure that you have created an integration project and message set project, and that you have imported an IDL file, as described in "Connecting to an external CORBA application" on page 1162.

About this task

You can create and configure a message flow manually, or you can create a message flow by dragging an imported IDL file onto the canvas.

Creating a message flow from an imported IDL file

Procedure

1. Drag an IDL file from the `CORBA IDLs` folder in the Application Development view to an empty canvas.

   (If you have imported an IDL file that contains includes, drag the top-level IDL file onto the canvas.)

   A CORBARequest node is created. The `IDL file`, `Interface name`, and `Operation name` properties are set according to the IDL file.
2. If the IDL file contains more than one interface or operation, select an interface and operation from the dialog box.
3. Configure the following properties on the CORBARequest node:
   - `Naming service`: Specify the host name and port of the naming service.

     The format of this value is host:port, where port is optional; for example, localhost:2809. You can obtain this value from the administrator of the CORBA application that you are calling.
   - `Object reference name`: Specify the name of the object reference in the naming service.

     You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 1159.
4. Add to the message flow other nodes that build the incoming and outgoing messages.

   You can use an XML message for the CORBARequest node, or you can build a message by using a Compute, or JavaCompute node. If the incoming message has a message model, you can use a Mapping node to build the message that is sent to the CORBARequest node.

   If the message that the CORBARequest node produces has a message model, you can use a Mapping node to build the outgoing message.
5. Build a message for the CORBARequest node by using the examples in "Building a message for the CORBARequest node" on page 1164.
6. Save the message flow.
7. Deploy the message flow.

   (If you have used an IDL file that contains includes, ensure that all the IDL files are deployed with the message flow.)

What to do next

You can also drag an IDL file onto an existing CORBARequest node. The existing `IDL file`, `Interface name`, and `Operation name` properties are replaced with values from the new IDL file, and the `Naming service` and `Object reference name` properties are cleared. If the IDL file contains more than one interface or operation, the `Interface name` property is set to the first interface in the IDL file, and the `Operation name` property is set to the first operation in that interface.

Creating a message flow manually

Procedure

1. Create a message flow.
2. Add a CORBARequest node to the message flow.
3. Configure the following properties on the CORBARequest node:
   - `Naming service`: Specify the host name and port of the naming service.

     The format of this value is host:port, where port is optional; for example, localhost:2809. You can obtain this value from the administrator of the CORBA application that you are calling.
   - `Object reference name`: Specify the name of the object reference in the naming service.

     You can obtain this value from the CORBA server that you are calling. For more information about how to specify the object reference name, see "CORBA naming service" on page 1159.
   - `IDL file`: Click **Browse** and select the IDL file from the message set project. If you have imported an IDL file that contains includes, select the top-level IDL file.
   - `Interface name`: Specify the name of the interface in the IDL file that the node calls.
   - `Operation name`: Specify the name of the operation from the interface that you select in the IDL file.

     You can override this property in the local environment by specifying a value in the following location:

     ```
     $LocalEnvironment/Destination/CORBA/Request/OperationName
     ```
4. Add to the message flow other nodes that build the incoming and outgoing messages.

   You can use an XML message for the CORBARequest node, or you can build a message by using a Compute, or JavaCompute node. If the incoming message has a message model, you can use a Mapping node to build the message that is sent to the CORBARequest node.

   If the message that the CORBARequest node produces has a message model, you can use a Mapping node to build the outgoing message.
5. Build a message for the CORBARequest node by using the examples in "Building a message for the CORBARequest node" on page 1164.
6. Save the message flow.
7. Deploy the message flow.

   (If you have used an IDL file that contains includes, ensure that all the IDL files are deployed with the message flow.)

What to do next

After you have deployed the message flow, learn how calls are processed by the CORBARequest node; see "Processing responses from a CORBARequest node" on page 1167.

Building a message for the CORBARequest node

You can use an XML message for the CORBARequest node, or you can build a message by using another message flow node.

Before you begin

Ensure that you have created and configured a message flow with a CORBARequest node, as described in "Developing a message flow with a CORBARequest node" on page 1163.

About this task

The CORBARequest node requires an input message. The node can use an XML message from another node, or you can build a message by using a Compute or JavaCompute node. If the incoming message has

a message model, you can use a Mapping node to build the message to send to the CORBARequest node. The incoming message tree must be in the DataObject domain, and the elements in the tree must match the IDL interface that you are calling. The physical representation of a message in the DataObject domain is XML; therefore, your application constructs message bodies like the following examples.

You can specify the location in the incoming message tree from which data is retrieved to form the request that is sent by the CORBARequest node. Specify this location by using the `Data location` property on the Request tab. The default value is `$Body`.

Procedure

1. Find out what data needs to be in the body of the message to send to the CORBARequest node.
2. Specify the top-level element (interface.operationName) for the message that you need to send to the CORBARequest node.
3. Specify a child for each in and inout parameter. You do not have to pass in a value for out parameters.

Example

The following examples show the XML and ESQL that could be received by a message flow.

A mixture of in inout and out parameters

Here is an example IDL file:

```
interface

ExampleOne {
 enum completion{YES, NO, MAYBE};
  typedef sequence<string> stringlist;
 struct stringobject {string member;};

 string exampleOneOperation(in string inparamA, inout string inoutparamA, out string outparamA);
 completion exampleOneOperationB(in stringlist inparamB, inout stringobject inoutparamB, out
 completion outparamB);
}
```

This IDL file contains an interface with two operations, which have an in parameter, an inout parameter, and an out parameter.

For the first operation, *exampleOneOperationA*, you must pass in the parameters *inparamA* and *inoutparamA* under the top-level type *ExampleOne.exampleOneOperationA*. You do not need to pass in the *outparamA* parameter.

Here is an XML example:

```
<ExampleOne.exampleOneOperationA>
 <inparamA>your value</inparamA>
 <inoutparamA>your value</inoutparamA>
</ExampleOne.exampleOneOperationA>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationA".inparamA = 'yourvalue';

SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationA".inoutparamA = 'yourvalue';
```

For the second operation, *exampleOneOperationB*, you must pass in the parameters *inparamB* and *inoutparamB* under the top-level type *ExampleOne.exampleOneOperationB*. You do not need to pass in the *outparamB* parameter.

Here is an XML example:

```
<ExampleOne.exampleOneOperationB>
 <inparamB><item>your value</item></inparamB>
 <inoutparamB><member>your value</member></inoutparamB>
</ExampleOne.exampleOneOperationB>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationB".inparamB.item = 'your value';

SET
OutputRoot.DataObject."ExampleOne.exampleOneOperationB".inoutparamB.member = 'your value'
```

In and inout parameters only

Here is an example IDL file:

```
interface

ExampleTwo {
 enum completion{YES, NO, MAYBE};
 typedef sequence<string> stringlist;
 struct stringobject {string member;};

 string exampleTwoOperationA(in string inparamA, inout string inoutparamA);
 completion exampleTwoOperationB(in stringlist inparamB, inout stringobject inoutparamB);

}
```

This IDL file contains two operations with an in and inout parameter only. The removal of the out parameter and the existence of only in and inout parameters does not change the parameters that you need to pass in.

To call the first operation, *exampleTwoOperationA*, pass in *inparamA* and *inoutparamA* under the top-level type *ExampleTwo.exampleTwoOperationA*.

Here is an XML example:

```
<ExampleTwo.exampleTwoOperationA>
 <inparamA>your value</inparamA>
 <inoutparamA>your value</inoutparamA>
</ExampleTwo.exampleTwoOperationA>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationA".inparamA = 'yourvalue';

SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationA".inoutparamA = 'yourvalue';
```

To call the second operation, *exampleTwoOperationB*, pass in *inparamB* and *inoutparamB* under the top-level type *ExampleTwo.exampleTwoOperationB*.

Here is an XML example:

```
<ExampleTwo.exampleTwoOperationB>
 <inparamB><item>your value</item></inparamB>
 <inoutparamB><member>your value</member></inoutparamB>
</ExampleTwo.exampleTwoOperationB>
```

Here is an ESQL example:

```
SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationB".inparamB.item = 'your value';

SET
OutputRoot.DataObject."ExampleTwo.exampleTwoOperationB".inoutparamB.member = 'your value';
```

No parameters or out parameters only

If the operation contains no parameters, or out parameters only, you do not need to put anything in the body of the message. The CORBARequest node does not look at the incoming message.

What to do next

Process the responses from the CORBARequest, as described in "Processing responses from a CORBARequest node" on page 1167.

Processing responses from a CORBARequest node

Configure the CORBARequest node to define the location to which responses are sent.

Before you begin

Complete the tasks that are described in the following topics:

- "Developing a message flow with a CORBARequest node" on page 1163
- "Building a message for the CORBARequest node" on page 1164

About this task

The CORBARequest node has three output terminals:

- Out: For a successful invocation of the operation.
- Error: If a CORBA system exception or user-defined exception is issued.
- Failure: If a failure occurs in the node; for example, if the node cannot communicate with the naming service or it receives an incorrect message.

You can select the location to which to send the response by configuring the `Output data location` property on the **Result** tab. This property specifies the message tree location to which the CORBARequest node puts output. The output is put under the DataObject domain. For more information, see "Combining a result message with an input message when fetching data from external systems" on page 1811.

Procedure

- **Processing successful calls**

  When a call is successful, the resulting message is propagated to the Out terminal. Inside the tree is a top-level element named *InterfaceName.OperationName*Response. Under this type is the return type of the operation (named _return) and every inout or out parameter. In parameters are not propagated because they do not change.

  A mixture of in inout and out parameters

  Here is an example IDL file:

  ```
  interface

  ExampleOne {
     string exampleOneOperation(in string inparam, out string outparam, inout string
   inoutparam);

  }
  ```

  You receive the parameters *outparam* and *inoutparam* under the top-level element `ExampleOne.exampleOneOperationResponse`.

  Here is an XML example:

  ```
  <ExampleOne.exampleOneOperationResponse>
   <_return>The operation return value</_return>
   <outparam>value from corba app</outparam>
   </inoutparam>your value changed by the corba app<inoutparam>
  </ExampleOne.exampleOneOperationResponse>
  ```

  Out and inout parameters

  Here is an example IDL file:

  ```
  interface
  ```

```
ExampleTwo {
   string exampleTwoOperation(out string outparam, inout string inoutparam);

}
```

The removal of the in parameter makes no difference to the message that is propagated from a CORBARequest node because in parameters are not propagated.

Here is an XML example:

```
<ExampleTwo.exampleTwoOperationResponse>
 <_return>The operation return value</_return>
 <outparam>value from corba app</outparam>
 </inoutparam>your value changed by the corba app<inoutparam>
</ExampleTwo.exampleTwoOperationResponse>
```

In parameters only or no parameters

If the operation contains no parameters, or in parameters only, you receive the _return value under the top-level element.

Here is an example IDL file:

```
interface

ExampleThree {
   string exampleThreeOperation(in string inparam);

}
```

Here is an XML example:

```
<ExampleThree.exampleThreeOperationResponse>
 <_return>The operation return value</_return>
</ExampleThree.exampleThreeOperationResponse>
```

No return type (void) and no inout or out parameters

If the operation that you are calling has no return type (void) and no inout or out parameters, no values are returned from the CORBA application to put in the tree. In this case, only the top-level element is created.

Here is an example IDL file:

```
interface

ExampleFour {
   void exampleFourOperation(in string inparam);

}
```

Here is an XML example:

```
<ExampleFour.exampleFourOperationResponse>
</ExampleFour.exampleFourOperationResponse>
```

- **Processing user-defined exceptions and CORBA system exceptions**

  When a CORBA user-defined exception or a CORBA system exception occurs, a message is propagated to the Error terminal. CORBA system exceptions have a standard shape and look like this:

  In the IDL file:

```
exception SystemException { // descriptive of error

 unsigned long;      // more detail about error
 CompletionStatus;   // yes, no, maybe
}
```

  In the tree:

```
<SystemException>
```

```
 <minor>10</minor>
 <completed>maybe</completed>
</SystemException>
```

The top-level element is the name of the CORBA system exception that is issued. The structure of a user-defined exception is based on the IDL for the exception. Here is an example:

In the IDL file:

```
  exception BadRecord {
 string why;

};
```

In the message:

```
<BadRecord>
 <why>reason text</why>
</BadRecord>
```

As you can see from this example, the structure of the message is based on the exception, and is not qualified by the operation that was being called.

- **Handling failures in the node**

  Any failures are propagated to the Failure terminal. Possible failures include:

  – Inability to communicate with the CORBA server
  – Inability to communicate with the CORBA naming service
  – Invalid body (including the wrong top-level element or missing parameters)

Defining where the CORBARequest node gets the object reference
You can specify an object reference name either on the CORBARequest node or by using a policy.

Before you begin

- Read “Common Object Request Broker Architecture (CORBA)” on page 1153 for background information.

About this task

By using policies, you can specify the location from which the CORBARequest node gets the object reference without the need to redeploy the message flow. You can also use the policy to specify this location for multiple CORBARequest nodes.

Creating and changing policies

Procedure

- To create or update a policy, use the Policy editor in the IBM App Connect Enterprise Toolkit (see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327).

Working with CICS Transaction Server for z/OS

Use the CICSRequest node to connect to CICS Transaction Server for z/OS applications.

About this task

This section contains the following concept information:

- “CICS Transaction Server for z/OS overview” on page 1170
- “CICS Transaction Server for z/OS connectivity” on page 1175
- “CICS Transaction Server for z/OS two-tier connectivity” on page 1177

- “CICS Transaction Server for z/OS three-tier connectivity” on page 1181
- “COMMAREA or channel data structures” on page 1171
- “CICS Transaction Server for z/OS mirror transactions” on page 1183
- “Local environment overrides for the CICSRequest node” on page 1185

This section contains the following tasks:

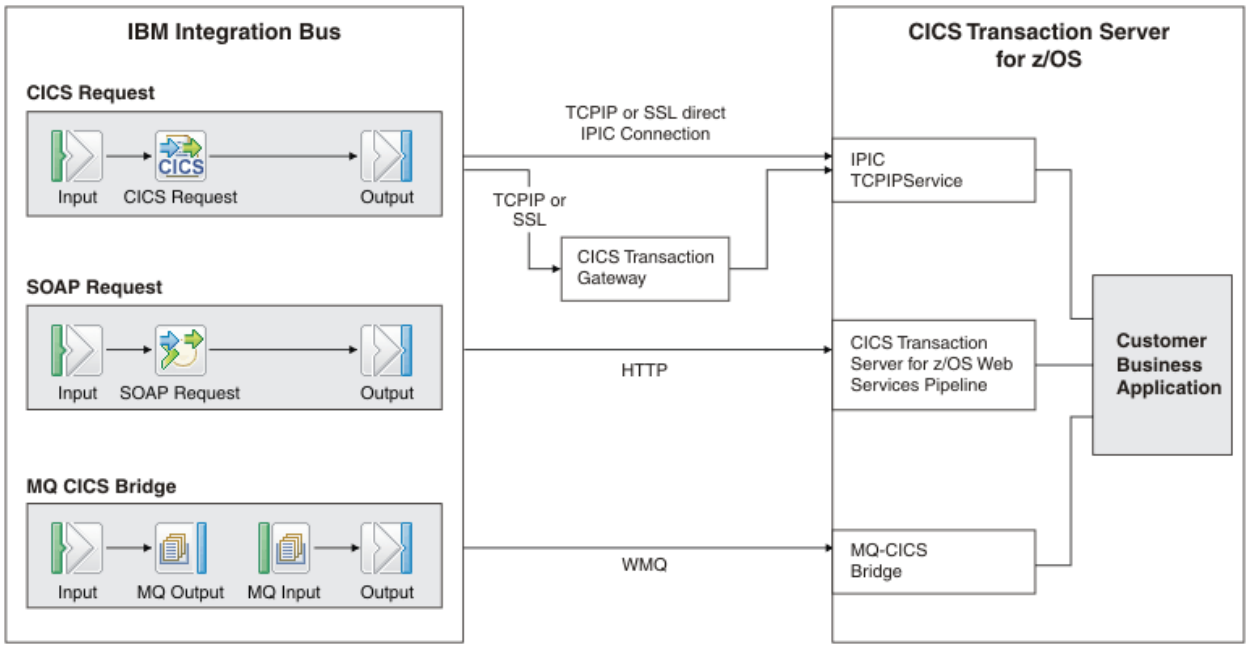
- “Connecting to a CICS Transaction Server for z/OS application” on page 1186
- “Defining a CICS Transaction Server for z/OS data structure” on page 1186
- “Preparing the environment for the CICSRequest node” on page 1187
- “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 1182
- “Developing a message flow with a CICSRequest node” on page 1188
- “Building a message for the CICSRequest node” on page 1191
- “Processing responses from a CICSRequest node” on page 1193
- “Changing connection information for the CICSRequest node” on page 1194
- “Propagating security credentials to CICS Transaction Server for z/OS” on page 1195

CICS Transaction Server for z/OS overview

CICS Transaction Server for z/OS provides general-purpose transaction processing software for z/OS. CICS is a powerful application server that meets the transaction-processing needs of both large and small enterprises.

By using the CICS support that is provided in IBM App Connect Enterprise you can deploy CICS applications into a service-oriented architecture (SOA). This support keeps your business logic intact and requires little or no change to CICS and CICS applications.

The following diagram shows the layers of communication between IBM App Connect Enterprise and CICS.



You can use the following methods to connect to CICS:

- **CICSRequest node support**

  IBM App Connect Enterprise includes a CICSRequest node that supports connectivity to CICS by using the IP InterCommunications (IPIC) protocol, which is available in CICS Transaction Server for z/OS.

To review the latest supported version of CICS Transaction Server for z/OS, see IBM App Connect Enterprise system requirements

Support includes the following connection methods:

– A direct connection to CICS (two-tier).

– A connection to CICS through CICS Transaction Gateway (three-tier).

- **CICS TG**

  CICS TG is a high-performing, secure, and scalable solution that provides workload management and high-availability options for access to CICS. By using standards-based interfaces, CICS TG deliver access to new and existing CICS applications.

For further information about the CICSRequest node, see CICSRequest node.

- **CICS web services support**

  You can use the web services support in IBM App Connect Enterprise to connect to CICS applications. For further information about web services, see “Processing web service messages” on page 805.

- **CICS-IBM MQ Integration**

  CICS has several components that support integration with IBM MQ. These components are:

  – The CICS-IBM MQ adapter, also known as the CICS-MQ attach (including the CKTI trigger monitor or task initiator). The term CICS-MQ adapter is used to refer to the CICS-IBM MQ adapter.

  – The CICS-IBM MQ bridge, also known as the CICS 3270 and DPL bridges. The term CICS-MQ bridge is used to refer to the CICS-IBM MQ bridge.

For further information about how to use the CICSRequest node to connect IBM App Connect Enterprise to CICS applications, see “CICS Transaction Server for z/OS connectivity” on page 1175.

For further information about CICS, see the CICS Library web page, which contains links to the CICS Information Center and associated IBM Redbooks publications.

COMMAREA or channel data structures

CICS Transaction Server for z/OS programs can be linked to by using either a COMMAREA data structure or a channel data structure as input, which return the same data structure as output. The CICSRequest node supports interaction with CICS through COMMAREA or channel data structures.

Channels are a modern alternative for COMMAREAs, providing relief from the COMMAREA maximum size of 32766 bytes, and allowing greater flexibility in input/output data structures.

- “COMMAREAs” on page 1171
- “Channels” on page 1172

COMMAREAs

If using a COMMAREA as the input data structure for communicating with CICS, the CICSRequest node takes a portion of the Input Body, as defined in the CICSRequest node Request properties, and sends it to CICS as the COMMAREA.

The returning COMMAREA is then put into the Output tree and replaces the existing Body at the location that is defined in the CICSRequest node Result properties. The COMMAREA can then be configured for parsing by using the CICSRequest node Response Message Parsing properties.

When defining a COMMAREA data structure as input, you must ensure that the CICSRequest node `Commarea length` property value is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of 32767 bytes. If the `Commarea length` value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS. The size of the COMMAREA cannot be changed by the CICS program. If the serialized request data is larger than the `Commarea length`, the data is truncated to the `Commarea length`. You can obtain the `Commarea length` value from the CICS administrator or developer.

The default value for the CICSRequest node `data structure` basic property is `Commarea`.

If using direct two-tier connection, CICS looks at the JVM `file.encoding` property to determine the code page of the COMMAREA. This can cause code page conversion errors in the CICS program. If the property is not set, a default value of 437 is used. For more information, see the CICS documentation about code pages.

For more information about using a COMMAREA data structure as input, see "Defining a CICS Transaction Server for z/OS data structure" on page 1186, "Developing a message flow with a CICSRequest node" on page 1188, and "Building a message for the CICSRequest node" on page 1191.

Channels

CICS channels hold a number of structures called containers. Containers hold the business information that is accessed by the target CICS program. Each container can hold up to 2 GB of data, and channels can have as many containers within them as required, which provides flexibility in terms of the size and layout of data. Each container has a 16-character maximum alphanumeric container name, which is unique within the channel, and is used as the mechanism to retrieve the contents of the container from the channel.

There are two types of container; `character` or `binary`. The type of container can affect the data conversion between IBM App Connect Enterprise and CICS, however the container type does not have any impact on the format of the information that can be put into the container.

Character containers

Character containers are more likely to be individual strings or discrete data items, but can also be mapped structures, however it is important to remember that data conversion is applied to the data in the container. When a character container is constructed, the source coded character set ID (CCSID) information about the container is sent to CICS as metadata. The CICS program uses the `GET CONTAINER` application programming interface (API) call to convert the metadata to the default CCSID of the region, unless another CCSID is provided. The CICS program then places the container back into the channel, and the data is converted ready for the IBM App Connect Enterprise application to retrieve and use.

Binary containers

Binary containers can be mapped by using a COBOL copybook structure or they might be discrete values. Data conversion is not applied to the data in a binary container, therefore the data in a binary container is sent to CICS and retrieved from CICS in original form only.

Unlike COMMAREA structures, the size of the response channel does not need to resemble the request, whereas COMMAREAs must allow for the size of the response in the request.

Channels and containers in CICS

In the following example diagram, the CICS channel has two containers; `CustomerName` and `Order`.

Channel Name:		MyChannel		
CustomerName	CHAR	'Joe Bloggs'		
Order	BIN	100	Apples	0.39
...	...	...		

`CustomerName` is a character (CHAR) container that contains a single character string; `Joe Bloggs`. Because `CustomerName` is a character container, data conversion can be applied to the data in the container. `Order` is a binary (BIN) container that might be created by using a COBOL copybook structure or C header file, which you can then import to populate your message set with message definitions. The following example copybook describes the binary layout of the data that the CICS program expects to receive:

```
01 ORDER_STRUCTURE.
          03 QTY                  COMP-1.
          03 ITEM                 PIC X(10).
```

```
           03 PRICE                PIC S9(9).
```

The target CICS program can retrieve both of these containers from the channel by providing the name of the container when using the GET CONTAINER API. When the data is provided to the CICS program, the program processes the data however it chooses. For example, the program can place other containers into the channel to provide a response to the called container by using the PUT CONTAINER API.

Channels and containers in IBM App Connect Enterprise

In IBM App Connect Enterprise, a CICS channel is represented as a message collection structure. A message collection can hold child messages, each treated as a container by the CICSRequest node. A message collection structure is used as both input and output to the CICSRequest node when using a channel data structure. For more information about message collections, see "Message collections" on page 1841, and for information about creating a message collection, see "Creating a message collection by using ESQL" on page 1843.

The message collection name is used to name the channel. The name of the child message in the message collection is used as the name of the container in the channel, and must be unique. If the child message name in the message collection is not unique, the request is rejected in CICS.

The following table shows the channel and container to message collection and child message mapping:

CICS	IBM App Connect Enterprise
Channel name	Message collection name
Container name (must be unique to the parent channel)	Child message name (must be unique to the message collection)

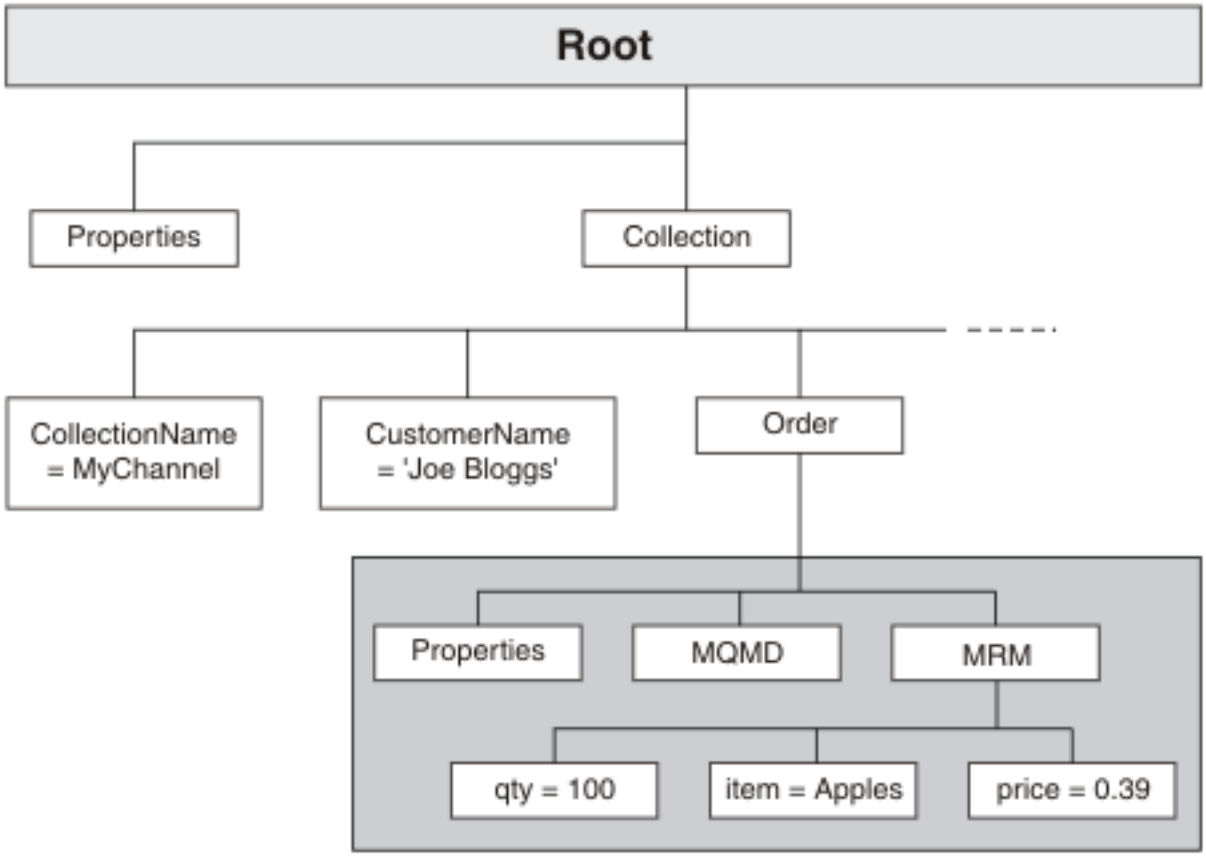
Name-value attributes

IBM App Connect Enterprise supports adding name-value attributes to a message collection to create a container. A message collection can have zero or more attributes. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called *CollectionName*.

You can add name-value attributes to a message collection to create CICS containers. Name-value attributes in the message collection, apart from *CollectionName*, can be used in lieu of full message-folders for simple data. For example, a name-value string attribute can be set in the message collection and used directly by the CICSRequest node without needing to create a message set for the element.

Name-value attributes can be produced from containers on output, as well as accepted for input. For information about creating an attribute instead of a message folder from a container, see CICSRequest node.

In the following example diagram, the CICS channel is represented by a message collection named Collection. Collection holds two containers that are represented by child messages named CustomerName and Order. *CollectionName* and CustomerName are both name-value attributes, however the *CollectionName* attribute is not treated as a container by the CICSRequest node, and is therefore not sent to CICS.



If the `CustomerName` attribute is to be treated as a character container by the CICSRequest node, the LocalEnvironment must reflect this.

LocalEnvironment

Each child message in a message collection is treated as having a default type of binary, which determines whether the data is converted to the CCSID in the CICS region. However, you can dynamically override this value to character, on a per message basis, in the local environment. For example, you can set the following value under `LocalEnvironment.Destination.CICS.RequestChannel.Containers`:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.Containers.<myContainerName> =
 CHARACTER;
```

When a message collection is emitted from the CICSRequest node following the request, the LocalEnvironment contains the returned type information for the containers. For example, when the response channel comes back from CICS, the LocalEnvironment shows the types of the containers that have come back in the following location: `LocalEnvironment.CICS.ResponseChannel.Containers.<myContainerName> = CHARACTER`

The channel name, which has a 16-character alphanumeric limit, can be overridden as follows:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.ChannelName = <myNewChannelName>;
```

If a single container is required for input only, a message collection does not need to be constructed. Instead a regular message can be used, however, you must set the 16-character maximum alphanumeric `ChannelName` in the LocalEnvironment. The 16-character maximum alphanumeric `SingleMessageContainerName` that needs to be created must also be provided in the following location:

```
SET OutputLocalEnvironment.Destination.CICS.RequestChannel.SingleMessageContainerName =
 <mySingleMessageContainerName>;
```

Because a message collection allows each container in the channel to be modeled as a separate message, each message has its own structure and parsing options. For example, one container might be XML and another might be based on a copybook, which can be represented by using XMLNSC and MRM messages within a message collection.

Each child message in the message collection contains message domain, set, type, format, CCSID, and encoding information in the `Properties` folder that is associated with the child message, which is serialized into a byte stream and sent to CICS. Each child message folder within the message collection that is being sent to CICS is serialized at the level of the last child of the message property domain. Not all CICS containers require a message set to represent them.

In the previous example, the `Order` container can be represented as MRM, and a message set can be created from the copybook ORDER_STRUCTURE to represent it. The returning channel is converted into a message collection, where every child message in the message collection represents a container from the channel. Child messages in the message collection are mapped to a list of message domain, set, type, format, CCSID, and encoding information by using the child message name, however CCSID, and encoding information are ignored for character messages. If a mapping cannot be found in the message, a default mapping can be provided.

Because it is not possible to know how many containers are in the response, a message collection is always produced as output.

You can use the CICSRequest node Response Message Parsing properties to map a returning container to a message domain, set, type, format, CCSID, and encoding information. In particular, the `Result data location` property can be used to reduce the result tree down to a single message folder, or down to a single field or subtree for output. For information about the `Result data location` property, see CICSRequest node.

The following details must be specified for the message to arrive at the CICSRequest node input terminal and be processed into a series of containers, and for those containers in the response channel to be placed back in the message as it leaves the node:

- The channel must be given a 16 character maximum alphanumeric name.

  Because the channel is represented by a message collection in IBM App Connect Enterprise, you can create the channel name by setting the message collection name. Message collection names are set by using the *`CollectionName`* attribute. For more information about creating a message collection and setting the message collection name, see “Creating a message collection by using ESQL” on page 1843.
- The following details must be specified for each container in the channel:
  - A 16 character maximum alphanumeric name.

    Because a container is represented by a child message in IBM App Connect Enterprise, you can create a container name by setting the child message name. For more information about creating a message collection and setting the child message name, see “Creating a message collection by using ESQL” on page 1843.
  - A container type; for example, `binary` or `character`.
  - A directory to use to place the response data into.

CICS Transaction Server for z/OS connectivity
Use the CICSRequest node to connect IBM App Connect Enterprise with CICS Transaction Server for z/OS applications.

The CICSRequest node that is available in IBM App Connect Enterprise provides connectivity to CICS applications by using IP InterCommunications (IPIC) protocol. IPIC is part of a CICS multi-version initiative to provide communications support over TCP/IP as an alternative to that provided over intersystem communication (ISC) and multiregion operation (MRO).

IPIC supports Distributed Program Link (DPL) requests over TCP/IP. The CICSRequest node communicates with CICS by sending Distributed Program Link (DPL) requests over TCP/IP-based IPIC. IPIC provides a multiplexed single socket connection to CICS Transaction Server for z/OS.

IBM App Connect Enterprise message flows can use the CICSRequest node to call programs that are running externally in a targeted CICS region. The CICSRequest node can be used by a message flow deployed to any integration node platform.

The CICSRequest node supports the following capabilities:

- Commareas
- Channels and containers
- Mirror transactions
- A direct connection to CICS (two-tier)
- A connection to CICS through CICS Transaction Gateway (three-tier)
- Local environment overrides for some properties
- Request timeout
- An integration node APPLID (client APPLID and qualifier) that can identify the application that is connecting to CICS, therefore identifying that IBM App Connect Enterprise is connected
- A CICS Connection policy
- Secure Sockets Layer (SSL) protocol connection to CICS
- Security identity
- User name, or user name and password identity propagation
- Resource statistics
- Transactionality

You can specify either a COMMAREA data structure or a channel data structure on the CICSRequest node to use as input for linking to CICS programs. The data structure that is specified as input returns the same data structure as output. Channels are an alternative for COMMAREAs, providing relief from the COMMAREA maximum size of 32766 bytes, and allowing greater flexibility in input/output data structures. For more information about using a COMMAREA or channel data structure, see "COMMAREA or channel data structures" on page 1171.

CICS channels hold a number of structures called containers. In IBM App Connect Enterprise, a CICS channel is represented as a message collection structure. A message collection can hold child messages, each treated as a container by the CICSRequest node. For information about using ESQL to create a message collection, see "Creating a message collection by using ESQL" on page 1843.

If a single container is required for input only, a message collection does not need to be constructed. Instead a regular message can be used, provided the 16-character maximum alphanumeric channel name and the single 16-character maximum alphanumeric container name are specified in the local environment. For more information about using single message mode, see "COMMAREA or channel data structures" on page 1171.

Because it is not possible to know how many containers are in the response, a message collection is always produced as output. However, the CICSRequest node `Result data location` property can be used to reduce the result tree down to a single message folder, or down to a single field or subtree for output. For information about the `Result data location` property, see CICSRequest node.

You can add name-value attributes to a message collection to create CICS containers. Name-value attributes in the message collection, apart from *`CollectionName`*, can be used in lieu of full message-folders for simple data. For example, a name-value string attribute can be set in the message collection and used directly by the CICSRequest node without needing to create a message set for the element. For more information about attributes, see "COMMAREA or channel data structures" on page 1171.

Name-value attributes can be produced from containers on output, as well as accepted for input. For information about creating an attribute instead of a message folder from a container, see CICSRequest node.

You can specify a mirror transaction name on the CICSRequest node for CICS tasks and programs to run under. This grouping greatly assists stat collection, accounting, and aids decision making about task

priority. For more information about mirror transactions, see “CICS Transaction Server for z/OS mirror transactions” on page 1183.

The CICSRequest node support in IBM App Connect Enterprise provides direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IPIC, or communication with CICS through CICS Transaction Gateway (three-tier connection). For more information about the two-tier and three-tier connection models, see “CICS Transaction Server for z/OS overview” on page 1170 for a high-level overview, or “CICS Transaction Server for z/OS two-tier connectivity” on page 1177 and “CICS Transaction Server for z/OS three-tier connectivity” on page 1181 for detailed conceptual information.

For information about configuring the CICSRequest node to get connection details from a CICS Connection policy, see “Changing connection information for the CICSRequest node” on page 1194.

You can configure the CICSRequest node or a CICS Connection policy to use SSL protocol. For more information, see “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 1182.

You can use the **mqsisetdbparms** command to set a user ID and password for the CICSRequest node or CICS Connection policy. For detailed information about how to configure CICS security identity support, see command.

The CICSRequest node can use an identity that is present on an input message, and propagate it to CICS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see “Propagating security credentials to CICS Transaction Server for z/OS” on page 1195 and “Identity and security token propagation” on page 2580.

You can use the CICSRequest node to connect to a CICS application by using a synchronous style of message flow. For details about how to use this node in a message flow, see CICSRequest node.

CICS Transaction Server for z/OS two-tier connectivity

The CICSRequest node support in IBM App Connect Enterprise provides direct communication with CICS Transaction Server for z/OS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol.

The CICSRequest node also supports communication with CICS through CICS Transaction Gateway (three-tier connection). For more information about three-tier connections, see “CICS Transaction Server for z/OS three-tier connectivity” on page 1181.

A direct two-tier connection from IBM App Connect Enterprise to CICS can be made by using the CICS Connection policy or by setting the properties directly on the CICSRequest node.

CICS Connection policy connections:

A CICS connection from IBM App Connect Enterprise is made to a listening TCPIPSERVICE resource in CICS. When that connection is established, the active connection between IBM App Connect Enterprise and CICS is represented by an IPCONN resource.

Each CICS Connection policy results in a separate connection to CICS, so for every policy that is being used, there is an IPCONN resource in CICS. The properties of the IPCONN resource determine the properties of the link between IBM App Connect Enterprise and CICS.

The IPCONN resource that represents an IBM App Connect Enterprise to CICS connection can be created in two different ways; autoinstall or pre-defined.

Autoinstall:

Autoinstalling a connection means that when IBM App Connect Enterprise connects, the resource is created, and when IBM App Connect Enterprise disconnects, the resource is discarded. In this setup, the IPCONN is created from a template IPCONN that is named by a user-replaceable-module (URM), which is named in the TCPIPSERVICE resource. Properties of the IPCONN are based on that template resource.

Pre-defined:

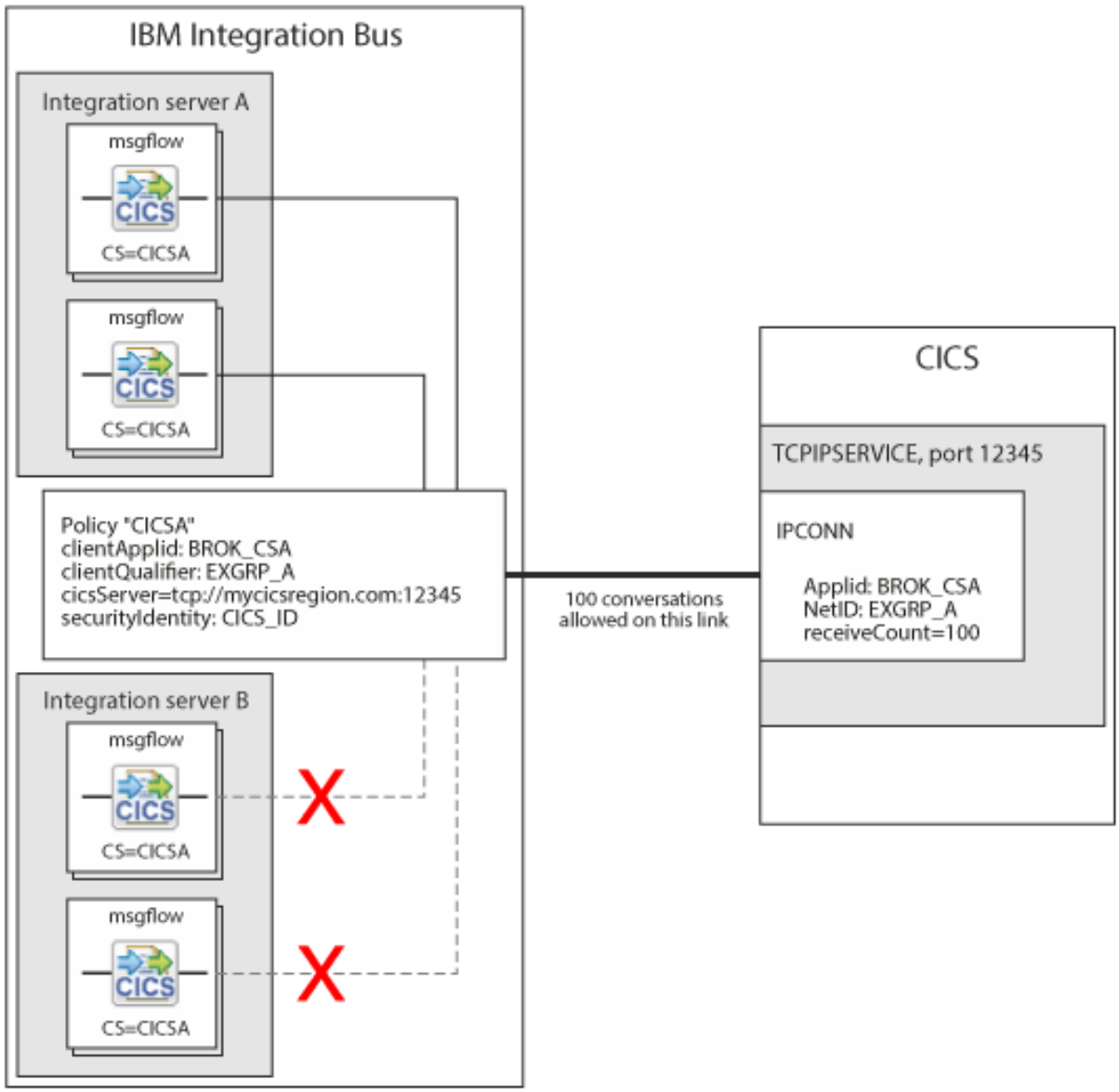
Alternatively, the IPCONN can be pre-defined by using standard CICS resource definition mechanisms, such as CICS Explorer, CEDA, or CICSPlex® Systems Manager (CICSPLEX SM). If the

IPCONN definition is created in advance, it is matched to the incoming connection by using the IPCONN APPLID and Network ID properties, which correlate to the `CICS APPLID` and `CICS APPLID qualifier` properties that can be set on a CICS Connection policy.

The advantage of pre-specifying the IPCONN is that you can tightly control the properties of incoming connections, including the security properties and the number of simultaneous requests. However the following rules apply:

- Do not configure different integration servers to use the same CICS Connection `CICS APPLID` and `CICS APPLID qualifier` combination to connect to the same CICS region. An IPCONN is tied to IBM App Connect Enterprise through the CICS Connection policy properties `CICS APPLID` and `CICS APPLID qualifier`. If this is attempted, only the first policy successfully connects.
- Do not specify a host name and port when defining the IPCONN resource in CICS. These fields are used for connections between CICS regions only, they must not be set for IBM App Connect Enterprise connections.

The following diagram shows how IBM App Connect Enterprise can directly connect to CICS by using a CICS Connection policy.



The two-tier direct CICS connection model is based on the following rules:

- Each policy name results in a separate connection to CICS.
- The CICS Connection policy must only be used from one integration server, because any further integration servers attempting to use the same policy thereafter fail to connect.
- The `CICS APPLID` and `CICS APPLID qualifier` properties in the policy are used to find the IPCONN resource in CICS. A chosen `CICS APPLID` and `CICS APPLID qualifier` combination must be unique to the CICS region. Only one IPCONN resource can exist with that combination.
- More than one message flow instance can use the CICS connection, however each request that goes through a CICSRequest node uses a conversation on the connection for the duration of the request.

When defining an IPCONN resource in CICS, consider the following properties:

- **CICS APPLID and Network ID**

  The CICS APPLID and Network ID properties must match the CICS Connection policy `clientApplid` and `clientQualifier` properties.

- **CICS host name and Port number**

  The CICS host name and port properties must be used for connections between CICS regions only, they must not be set for IBM App Connect Enterprise connections.

- **CICS TCPIPSERVICE**

  IPCONNs are owned by a parent TCPIPSERVICE resource in CICS.

- **CICS Receivecount**

  The CICS Receivecount property controls the number of simultaneous requests that can be performed over the connection. The number of simultaneous requests defaults to 100 for autoinstalled connections.

- **CICS Sendcount**

  The Sendcount property must be set to 0 because the Sendcount property is used for CICS connections only, and must not be used for IBM App Connect Enterprise connections.

- **CICS LINKAUTH**

  The CICS LINKAUTH property controls how the link security is managed. To use a resource in CICS, two security checks are performed; the "flowed" user, which checks the security credentials that are sent from IBM App Connect Enterprise, and the "link" user, which must also have permission for the resource. Both user IDs must have permission to use the resource before the request is granted. The link user ID is given low privileges, which means that even if the flowed user has many permissions, the link user ID can be used to cap the privilege of the connection. If LINKAUTH is set to SECUSER, the SECURITYNAME field is used to specify the link user ID. If set to CERTUSER, the link user is determined from an SSL client certificate that is mapped by RACF. If USERAUTH(IDENTIFY) or USERAUTH(VERIFY) is specified, the link user ID is not used. Only the user ID received from the TOR is used to determine security.

- **CICS USERAUTH**

  The CICS USERAUTH property determines how the flowed user security is configured. If USERAUTH is set to "LOCAL" or "DEFAULTUSER", no user ID or password is to be sent to CICS on a request. This means that all requests use the CICS region ID. If USERAUTH is set to "IDENTIFY", user IDs are flowed without a password. If USERAUTH is set to "VERIFY", user IDs and passwords are required. If USERAUTH(IDENTIFY) or USERAUTH(VERIFY) is specified, the link user ID is not used. Only the user ID received from the TOR is used to determine security.

Each CICSRequest node in a message flow acts as a request on one of the connections to CICS. Which connection is used is determined by the policy that is used.

For more information about configuring the CICSRequest node to get connection details from a CICS Connection policy, see “Changing connection information for the CICSRequest node” on page 1194.

You can configure the CICSRequest node or a CICS Connection policy to use SSL protocol. For more information, see “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 1182.

CICSRequest node connections:

If a CICS Connection policy is not specified on the CICSRequest node, and a host name is used directly in the `CICS server` property, the request shares a connection with other resources that have specified the same `CICS server` URL. The first CICSRequest node to be used opens the connection to CICS, regardless of whether a URL or a policy is specified in the `CICS server` property.

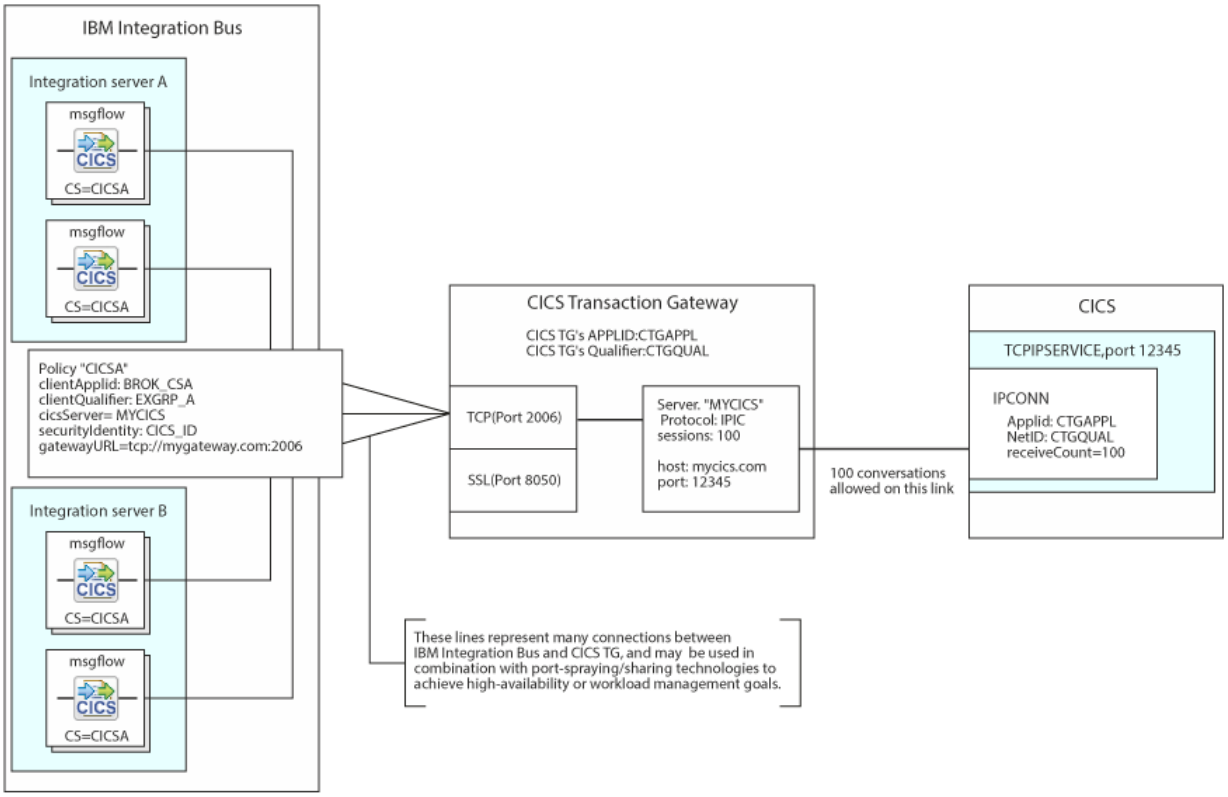
CICS Transaction Server for z/OS three-tier connectivity

The CICSRequest node support in IBM App Connect Enterprise can provide communication with CICS Transaction Server for z/OS through CICS TG (three-tier connection).

The CICSRequest node also supports direct communication with CICS (two-tier connection) by sending Distributed Program Link (DPL) requests over TCP/IP-based IP InterCommunications (IPIC) protocol. For more information about two-tier connections, see "CICS Transaction Server for z/OS two-tier connectivity" on page 1177.

A three-tier connection from IBM App Connect Enterprise to CICS through CICS TG can be made by configuring the `CICS Server` and `CICS Transaction Gateway URL` CICS Connection policy properties.

The following diagram shows how IBM App Connect Enterprise can make a connection to CICS through CICS TG by using the CICS Connection policy.



For more information about configuring the `CICS server` and `CICS Transaction Gateway URL` CICS Connection policy properties to make a three-tier connection, see CICS Connection policy (CICSConnection).

The three-tier connection to CICS through CICS TG connection model is based on the following rules:

- The number of connections that can be made to CICS TG is determined by the maximum number of simultaneous CICS requests in progress. Ensure that the Connection Manager and Worker Thread resources in your CICS TG deployment have sufficient capacity to handle the number of required IBM App Connect Enterprise connections.
- The `CICS APPLID` and `CICS APPLID qualifier` properties in the CICS Connection policy can be used to identify the IBM App Connect Enterprise connection within CICS TG. In addition, any CICS tasks that are started through the policy each contain point of origin information, including the specified client APPLID and qualifier, which you can find in the CICS task association data.
- Unlike two-tier connections, there is no restriction about sharing client APPLIDs between integration servers because the connection is made to CICS TG, and not to CICS directly.

For more information about configuring the CICSRequest node to get connection details from a CICS Connection policy, see "Changing connection information for the CICSRequest node" on page 1194.

Securing the connection to CICS Transaction Server for z/OS by using SSL
Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICS Connection policy or the CICSRequest node to use SSL.

Before you begin

Ensure that you have completed the following tasks:

1. The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. If client-authentication (CLIENTAUTH) is enabled in the TCPIPSERVICE in CICS, the IBM App Connect Enterprise keystore file must also contain a personal certificate that is trusted by CICS.
2. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 1186.
3. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 1187.

About this task

To configure the CICSRequest node to use SSL, complete the following steps:

Procedure

1. For client-authenticated (CLIENTAUTH) SSL connections, CICS expects the SSL client certificate to be mapped to a RACF user ID. Therefore the SSL client certificate must be mapped to a RACF user ID before attempting to establish the SSL connection to CICS. If the client certificate is not mapped to a RACF user ID, IBM App Connect Enterprise might display a `ECI_ERR_NO_CICS` response. You can map a client certificate to a RACF user ID by using the RACF command **RACDCERT**, which stores the client certificate in the RACF database and associates a user ID with it, or by using RACF certificate name filtering. Client certificates can be mapped one-to-one with a user ID, or a mapping from one to

the other can be provided to allow a many-to-one mapping. You can achieve this mapping by using one of the following methods:

- **Associating a client certificate with a RACF user ID**

  a. Copy the certificate that you want to process into an MVS™ sequential file. The file must have variable length, blocked records (RECFM=VB), and be accessible from TSO.

  b. Run the **RACDCERT** command in TSO by using the following syntax:

  ```
  RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
  ```

  Where:

  – *datasetname* is the name of the data set containing the client certificate.

  – *userid* is the user ID to be associated with the certificate. This parameter is optional. If omitted, the certificate is associated with the user issuing the **RACDCERT** command.

  When you issue the **RACDCERT** command, RACF creates a profile in the DIGTCERT class. This profile associates the certificate with the user ID. You can then use the profile to translate a certificate to a user ID without giving a password. For full details of RACF commands, see z/OS Security Server RACF Command Language Reference.

- **RACF certificate name filtering**

  With certificate name filtering, client certificates are not stored in the RACF database. The association between one or more certificates and a RACF user ID is achieved by defining a filter rule that matches the distinguished name of the certificate owner or issuer (CA). A sample filter rule might look like the following example:

  ```
  RACDCERT ID(DEPT3USR) MAP SDNFILTER
  (OU=DEPT1.OU=DEPT2.O=IBM.L=LOC.SP=NY.C=US)
  ```

  This sample filter rule would associate user ID DEPT3USR with all certificates when the distinguished name of the certificate owner contains the organizational unit DEPT1 and DEPT2, the organization IBM, the locality LOC, the state/province NY, and the country US.

2. Turn on SSL support in the integration server by setting the `CICS Server` property on the CICS Connection policy.

   Alternatively you can configure the `CICS server` property directly on the CICSRequest node.

What to do next

When you have configured the integration server or the CICSRequest node to use SSL, develop a message flow that contains a CICSRequest node by following the steps in “Developing a message flow with a CICSRequest node” on page 1188.

CICS Transaction Server for z/OS mirror transactions

You can use a mirror transaction to group CICS Transaction Server for z/OS tasks and programs together. This grouping greatly assists stat collection, accounting, and aids decision making about task priority.

You can specify a mirror transaction name on the CICSRequest node for CICS tasks and programs to run under. For example, if each department in a business has a different mirror transaction name, work can be tracked back to the correct source, and decisions about task priority and quality of service (QoS) can be made in CICS. Potentially, different security privileges might be available depending on the transaction name chosen. Or the transaction name might be used as a way of indicating the origin of the task. Alternatively, a mirror transaction can be used to denote whether code page translation of the commarea data is required.

There are two ways that a mirror transaction can be specified:

Specifying a mirror transaction name that corresponds with a defined TRANSACTION resource in CICS:

You can specify a mirror transaction name by configuring the `Mirror transaction ID` property on the CICSRequest node Basic tab, however the `Mirror transaction ID` property value that you specify must correspond to a defined TRANSACTION resource in CICS. For example, if you have a defined TRANSACTION resource of ATRN in CICS, and you want tasks and programs to run under that transaction, you must configure ATRN as the `Mirror transaction ID` property value.

When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction.

Specifying a weaker form of mirror transaction that does not require a TRANSACTION resource to be defined in CICS:

You can use a weaker form of mirror transaction that does not change the TRANSACTION resource, but instead sets a variable called `EIBTRNID`, which is available to the called program. You can configure the `EIBTRNID` variable to tell the program what TRANSACTION resource it is running under, without the TRANSACTION resource being defined in CICS.

For example, you can specify this weaker form of mirror transaction by configuring the `Mirror transaction ID` property with the name of the required TRANSACTION resource; for example ATRN, and by selecting the `Set EIBTRNID only` property on the CICSRequest node Basic tab.

When the message flow containing the configured CICSRequest node is deployed, any CICS programs that are started thereafter appear in CICS as running under the specified mirror transaction.

If the value of the CICSRequest node `Mirror transaction ID` property is not set, the mirror transaction name defaults to `CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system.

The following table describes the mirror transaction handling that is applied depending on the configuration of the `Mirror transaction ID` and `Set EIBTRNID only` CICSRequest node properties. Where ATRN is an example of a user-defined transaction name:

Mirror transaction ID property value	**Set EIBTRNID only property value**	**Task and programs run under defined TRANSACTION resource:**	**EIBTRNID is:**
Blank	Cleared	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system
ATRN	Cleared	ATRN	ATRN
ATRN	Selected	`CPMI` if called by a distributed platform, or `CSMI` if called by a z/OS system	ATRN

If you are considering whether to use a mirror transaction as a way of finding the point of origin of your data, using the CICS task association data might be a better alternative. All tasks that are initiated in CICS over IP InterCommunications (IPIC) protocol contain origin information, including source Internet Protocol (IP) and APPLID information.

The CICS mirror transaction properties can be changed by configuring the properties directly on the CICSRequest node, by using the **mqsiapplybaroverride** command, or by dynamically overriding these property values with elements in the message tree, on a per message basis, in the local environment. For more information about dynamically overriding CICSRequest node values, see "Local environment overrides for the CICSRequest node" on page 1185 and for information about the **mqsiapplybaroverride** command, see Configurable properties of message flow nodes.

Local environment overrides for the CICSRequest node
When you use the CICSRequest node in a message flow, you can override some of its properties with elements in the message tree.

You can dynamically override values, on a per message basis, in the local environment. You can override the CICS Transaction Server for z/OS program that you are calling, the COMMAREA length, mirror transactions, and the processing of the response message.

You can set the following values under `LocalEnvironment.Destination.CICS`.

Setting	Description
CICSProgramName	Overrides the `Program name` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.CICSProgramName = 'progx';` You can override the location of this value by using a property on the CICSRequest node.
CICSCommareaLen	Overrides the `Commarea length` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.CICSCommareaLen = 100;`
mirrorTran	Overrides the `Mirror transaction ID` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.mirrorTran = 'ATRN';`
eibtrnidOnly	Overrides the `Set EIBTRNID only` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.eibtrnidOnly = 'TRUE';` Valid values are TRUE, FALSE, YES, or NO. Where TRUE or YES represents a selected check box for the `Set EIBTRNID only` property in the IBM App Connect Enterprise Toolkit. Where FALSE or NO represents a cleared check box in the IBM App Connect Enterprise Toolkit.

You can set the following values under `LocalEnvironment.Destination.CICS.Response`.

Setting	Description
messageDomain	Overrides the `Message domain` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageDomain = 'MRM';`
messageSet	Overrides the `Message set` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageSet = '{com.test}:MyMessageSet';`
messageType	Overrides the `Message type` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageType = 'messageA';`
messageFormat	Overrides the `Message format` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageFormat = 'XML1';`
messageCCSID	Overrides the `Message coded character set ID` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageCCSID = 500;`

Setting	Description
messageEncoding	Overrides the `Message encoding` property on the node; for example: `SET OutputLocalEnvironment.Destination.CICS.Response.messageEncoding = 785;`

You can also set LocalEnvironment values for CICS channels and containers. For more information, see "COMMAREA or channel data structures" on page 1171.

Connecting to a CICS Transaction Server for z/OS application

Connecting to a CICS Transaction Server for z/OS application involves creating a message flow, building a message, and processing the response from the CICSRequest node.

About this task

To connect to a CICS application, complete the following steps.

Procedure

1. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 1186.
2. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 1187.
3. Optional: Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol, as described in "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 1182.
4. Develop a message flow that contains a CICSRequest node, as described in "Developing a message flow with a CICSRequest node" on page 1188.
5. Build a message to send to the CICSRequest node, as described in "Building a message for the CICSRequest node" on page 1191.
6. Process the response from the CICSRequest node, as described in "Processing responses from a CICSRequest node" on page 1193.

What to do next

You can change the connection details for the CICSRequest node by using the CICS Connection policy, as described in "Changing connection information for the CICSRequest node" on page 1194.

The CICSRequest node can also use an identity that is present on an input message, and propagate it to CICS, by using the `Propagate` property on the security profile that is defined for the node. For more information, see "Propagating security credentials to CICS Transaction Server for z/OS" on page 1195

Defining a CICS Transaction Server for z/OS data structure

Mapped data structures, such as COBOL copybooks or C structures, can be used to define an MRM message definition for a CICS Transaction Server for z/OS COMMAREA or a channel container.

Before you begin

- Optional: If you are using a COMMAREA data structure, read the COMMAREA concept information that is defined in the CICS Transaction Server for z/OS Version 4.1 product documentation online.
- Read the COMMAREA and channel concept information that is defined in "COMMAREA or channel data structures" on page 1171.

- Create a message definition file by using the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit.
  - If you are using a C header file, see "Message Sets: Importing from C" on page 2300.
  - If you are using a COBOL data structure, see "Message Sets: Importing from COBOL copybooks" on page 2302.
  - For information about other data structures, see "Message Sets: Working with data structures" on page 2299.

About this task

After you have created the message definition, complete the following steps.

Procedure

1. Select the `Data structure` property value on the **Basic** tab of the CICSRequest node in accordance with the targeted CICS program. For example, if the target program is channel-based, select `Channel` as the data structure.
2. Ensure that the properties on the **Response Message Parsing** tab of the CICSRequest node specify the output COMMAREA or container structure.
3. Check that the upstream nodes are configured to provide the correct input structure.
4. Optional: If you are using a COMMAREA data structure, ensure that the CICSRequest node `Commarea length` property is large enough to hold the serialized input or output structures to avoid a memory leak in the CICS application.

What to do next

If you are using a channel data structure, create a message collection to represent the channel data structure, as described in "Creating a message collection by using ESQL" on page 1843.

Preparing the environment for the CICSRequest node

Before you can use the CICSRequest node, you must configure IP InterCommunications (IPIC) protocol on the target CICS Transaction Server for z/OS.

Before you begin

Read "CICS Transaction Server for z/OS connectivity" on page 1175 for background information.

About this task

The CICSRequest node can send IPIC requests over TCP/IP to CICS Transaction Server for z/OS. Complete the following steps on CICS to perform this configuration:

Procedure

1. Set the System Initialization (SIT) parameter **TCPIP=YES**.
2. Define the TCP/IP address and host name for CICS. By default, they are defined in the PROFILE.TCPIP and TCPIP.DATA data sets.

3. Add a TCP/IP listener to CICS by using the following **CEDA** command to define a TCPIPSERVICE resource in a group:

```
CEDA DEF TCPIPSERVICE(service-name) GROUP(group-name)
```

Ensure that the group in which you define the service is in the GRPLIST system initialization parameter, so that the listener starts when CICS is started. Key fields are explained as follows:

POrtnumber:
The port on which the TCP/IP service listens.

PROtocol:
The protocol of the service is IPIC.

TRansaction:
The transaction that CICS runs to handle incoming IPIC requests. Set the field to CISS, which is the default.

Backlog:
The Backlog field is the number of TCP/IP connection requests are sent to CICS, which are placed in a TCP/IP queue to be assigned an IPCONN connection to CICS. The default value is 1. Do not use a value of 0; a value of 0 indicates that no TCP/IP connection requests are to be assigned an IPCONN connection to CICS, which disables incoming connection requests.

Ipaddress:
The IP address, in dotted decimal form, on which the TCPIPSERVICE resource listens. For configurations with more than one IP stack, specify ANY to make the TCPIPSERVICE resource listen on all addresses.

SOcketclose:
Whether CICS waits before closing the socket after issuing a receive for incoming data on that socket. To ensure that the connection from the CICSRequest node always remains open, set SOcketclose to NO for IPIC connections.

SSl:
Whether the CICS TCP/IP service is to use Secure Sockets Layer (SSL) protocol for encryption and authentication. Valid values are NO, YES, or CLIENTAUTH. Where:

- NO indicates that SSL is not to be used.
- YES indicates that the personal certificate of the CICS region must be trusted by IBM App Connect Enterprise.
- CLIENTAUTH indicates that the personal certificate of the CICS region must be trusted by IBM App Connect Enterprise, and the IBM App Connect Enterprise personal certificate must be trusted by CICS.

  The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. For more information, see “Securing the connection to CICS Transaction Server for z/OS by using SSL” on page 1182.

4. Use the following command to install the TCPIPSERVICE definition:

```
CEDA INS TCPIPSERVICE(service-name) GROUP(group-name)
```

Developing a message flow with a CICSRequest node
To connect to a CICS Transaction Server for z/OS application, create a message flow that contains a CICSRequest node.

Before you begin

Ensure that you have completed the following tasks:

1. Define the COMMAREA or channel data structure, as described in “Defining a CICS Transaction Server for z/OS data structure” on page 1186.

2. Optional: If you are using a channel data structure, create a message collection to represent the channel data structure, as described in "Creating a message collection by using ESQL" on page 1843.
3. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 1187.
4. Optional: Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol, as described in "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 1182.

About this task

Complete the following steps to develop a message flow with a CICSRequest node.

Procedure

1. Create a message flow.
2. Add a CICSRequest node to the message flow.
3. Configure the following properties on the CICSRequest node.
   - `CICS server`: The `CICS server` property can be defined either as a policy name, for example *myCICSConnection*, or as a URL.

     You can either connect to CICS by using the two-tier connection model; for example, by making a direct connection from IBM App Connect Enterprise to CICS, or by using the three-tier connection model; for example, by connecting to CICS through CICS Transaction Gateway. For more information about the two-tier and three-tier connection models, see "CICS Transaction Server for z/OS overview" on page 1170 for a high-level overview, or "CICS Transaction Server for z/OS two-tier

connectivity" on page 1177 and "CICS Transaction Server for z/OS three-tier connectivity" on page 1181 for detailed conceptual information.

Using the two-tier connection model:
If you are making a direct two-tier connection from IBM App Connect Enterprise to CICS, you can define the `CICS server` property either as a policy name, for example *myCICSConnection*, or as a URL.

For more information about defining this property as a policy, see "Changing connection information for the CICSRequest node" on page 1194.

To define a URL, specify the protocol and the CICS host name and port number. The format of this value is *protocol*`://`*hostname*`:`*port*. Where:

- *protocol* can be `tcp` or `ssl`.
- *hostname* is the TCP/IP address of the CICS host.
- *port* is the port number of the TCPIPSERVICE listener in CICS that is listening for IPIC requests over TCP/IP or SSL.

For example: `tcp://mycicsregion.com:12345` or `ssl://mycicsregion.com:56789`. You can obtain the *hostname* and *port* values from the IPIC TCPIPSERVICE definition in the target CICS region.

Using the three-tier connection model:
If you are making a three-tier connection to CICS through CICS Transaction Gateway, the `CICS server` CICSRequest node property must be defined as a policy name, for example *myCICSConnection*.

For more information about defining this property as a policy, see "Changing connection information for the CICSRequest node" on page 1194.

To make a three-tier connection to CICS through CICS Transaction Gateway, you must configure the `CICS Server` and `CICS Transaction Gateway URL` CICS Connection policy properties (see CICS Connection policy (CICSConnection)).

- `Program name`: Specify the name of the program that you want to run in the target CICS region.

  You can override this property in the local environment by specifying a value in the following location:

  ```
  $LocalEnvironment/Destination/CICS/CICSProgramName
  ```

- `Data structure`: Specify whether to use a COMMAREA or a channel data structure. The default for this property is `Commarea`. The decision depends on the targeted CICS program, for example; whether the target program is channel-based or not.
- `Commarea length`:

  This property is not configurable if a value of `Channel` is selected for the `Data structure` property.

  The `Commarea length` property is the size, in bytes, of the COMMAREA that is used by the CICS program. The byte size value is sent to CICS, and before the program is started, an area of memory is created to match that number. For example, if you send a `Commarea length` value of `100`, 100 bytes are allocated. The program accesses this area as the `DFHCOMMAREA`.

  Ensure that the `Commarea length` property value is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of 32767 bytes. If the

Commarea length value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS.

The size of the COMMAREA cannot be changed by the CICS program.

If the serialized request data is larger than the Commarea length, the data is truncated to the Commarea length.

You can obtain the Commarea length value from the CICS administrator or developer.

You can override this property in the local environment by specifying a value in the following location:

```
$LocalEnvironment/Destination/CICS/CICSCommareaLen
```

- Transaction mode: Specify whether requests to the CICSRequest node are to managed as transactional or non-transactional.
  - If you select Yes, the CICSRequest node takes part in the local transaction that is started by the message flow's input node.
  - If you select No, the CICSRequest node does not take part in the local transaction that is started by the message flow's input node.
  - If you select Automatic, the message transactionality is inherited from the Transaction mode setting on the Input node at the start of the message flow. For example, if the message flow is driven by an MQInput node, the CICSRequest node assumes the Transaction mode that is set on the MQInput node. By default, the Transaction mode property of the CICSRequest node is set to Automatic.

4. Save the message flow.
5. Deploy the message flow.

What to do next

When you have created and configured the message flow, build a message by following the steps in “Building a message for the CICSRequest node” on page 1191.

Building a message for the CICSRequest node

Create a message definition from a data structure and build a message by using another message flow node.

About this task

You can specify the location in the incoming message tree from which data is retrieved to form the request that is sent by the CICSRequest node. Specify this location by using the Data location Request property on the CICSRequest node. For more information, see “Combining a result message with an input message when fetching data from external systems” on page 1811.

Procedure

1. Find out what data needs to be in the body of the message to send to the CICSRequest node. The message body data must match the input structure that is required for your commarea, as defined in a message set definition based on the language structure. For example; COBOL or C copybook.
2. Ensure that the Commarea length Basic property value that is configured in the CICSRequest node is large enough to contain the input request data, or the output response data, but that it does not exceed the maximum value of 32767 bytes. If the Commarea length value is not large enough to be used for the response data, or the request data, a memory leak occurs in CICS. The size of the commarea cannot be changed by the CICS program. If the serialized request data is larger than the Commarea length, the data is truncated to the Commarea length. You can obtain the Commarea length value from the CICS administrator or developer.

Example

The following example shows a message that has been modeled in the MRM domain, which can be received by a CICSRequest node and sent to CICS.

COBOL copybook

This example shows the structure of the data that CICS is expecting. The copybook describes the binary layout of the data that the CICS program expects to receive.

```
01 DFHAXCS-REQUEST.
   10 AXCS-COMMAND                  PIC S9(9) COMP.
   10 AXCS-FILE                     PIC X(8).
   10 AXCS-RIFLD                    PIC X(6) VALUE SPACES.
   10 AXCS-DATA.
      15 AXCS-STAT                  PIC X(1)  VALUE SPACES.
      15 AXCS-RECID                 PIC X(6)  VALUE SPACES.
      15 AXCS-NAME                  PIC X(20) VALUE SPACES.
      15 AXCS-ADDRESS               PIC X(20) VALUE SPACES.
      15 AXCS-PHONE                 PIC X(8)  VALUE SPACES.
      15 AXCS-DATE                  PIC X(8)  VALUE SPACES.
      15 AXCS-AMOUNT                PIC X(8)  VALUE SPACES.
      15 AXCS-COMMENT               PIC X(9)  VALUE SPACES.
```

The example copybook can be used to create a binary structure that requires 98 bytes of COMMAREA or memory space, as shown in the following example:

Table 37.

Name	**Type and Size**
AXCS-COMMAND	4 byte integer (fullword)
AXCS-FILE	8 byte character string
AXCS-RIFLD	6 byte character string
AXCS-STAT	1 byte character string
AXCS-RECID	6 byte character string
AXCS-NAME	20 byte character string
AXCS-ADDRESS	20 byte character string
AXCS-PHONE	8 byte character string
AXCS-DATE	8 byte character string
AXCS-AMOUNT	8 byte character string
AXCS-COMMENT	9 byte character string
Total	**98 bytes**

The COBOL copybook structure must be imported as a message definition, see “Message Sets: Importing from COBOL copybooks” on page 2302 for further information, and a message containing such a structure must be passed to the CICSRequest node. A second copybook might be required to map the returning COMMAREA.

What to do next

Process the responses from the CICSRequest, as described in “Processing responses from a CICSRequest node” on page 1193.

Processing responses from a CICSRequest node
The CICSRequest node can return different response messages that indicate the success or failure of the request sent to CICS Transaction Server for z/OS.

Before you begin

Ensure that you have built a message for the CICSRequest node, as described in "Building a message for the CICSRequest node" on page 1191.

About this task

The CICSRequest node has four output terminals:

- Out: The output terminal from which the message tree is propagated, including the data returned from CICS.
- Failure: The output terminal to which a message is routed if a CICSRequest node exception is detected, or a CICSRequest node to CICS connection failure occurs.
- Error: The output terminal to which a message is propagated if a CICS error (abend) occurs.
- Timeout: The output terminal to which the message is propagated if a per-request timeout occurs when an individual request is sent to CICS, but the request takes too long.

You can select the location to which to send the response by configuring the `Output data location` Result property on the CICSRequest node. This property specifies the location in the message tree to which the CICSRequest node places the output.

Procedure

- **Processing successful calls**

  When a CICSRequest node successfully calls a CICS application, the resulting message is propagated to the Out terminal.

- **Processing CICS abends**

  An abend in CICS causes a message to be propagated from the Error terminal. The input message is propagated with a CICS\AbendCode field in the LocalEnvironment. If the Error terminal is not connected, the abend is lost.

  If an abend occurs and the CICSRequest node is configured to participate in the integration node transaction, all work carried out as part of the CICS task is immediately rolled back. If multiple CICS nodes are involved in the unit of work, the CICS activity of each of the nodes is rolled back, whether or not the abend is handled through an error or failure terminal. Any further work carried out by CICS as part of processing the message is carried out under a new unit of work.

  A unit of work is the work that is carried out by nodes in a flow that share the same resource; in this case, a unit of work is all the work that is carried out by CICSRequest nodes in a flow that are connected to the same CICS server (through a URL or policy defined in the **`CICS server`** property). For example, if a message flow contains five CICS nodes, three of which are connected to CICS server A and two of which are connected to CICS server B, all the work that is carried out by nodes connected to server A forms one unit of work, and all the work that is carried out by nodes connected to server B forms a separate unit of work. If an abend occurs in one of the CICS nodes that are connected to CICS server A, the work carried out by all the CICS nodes in the flow that are connected to server A is rolled back, because it forms a single unit of work. Any subsequent work that is carried out by a CICSRequest node connected to server A will start a new unit of work, and the unit of work for server B is unaffected by the rollback of the unit of work for server A.

- **Processing request timeouts**

  If a CICSRequest node is configured with a `Request timeout` Basic property, and a particular message takes longer than the specified time in seconds to be processed, the request fails and the message is propagated from the Timeout terminal. The output message contains the input message

body and a timeout exception in the ExceptionList. If the Timeout terminal is not connected and a timeout occurs, the request timeout exception is routed to the Failure terminal. If the Failure terminal is not connected, the integration node throws an exception and returns control to the closest upstream node that can process it. The default behavior is that the message is returned to the input node.

- **Handling failures in the CICS node**

  Any other failures are propagated to the Failure terminal. Possible failures include:

  – An inability to communicate with the target CICS region.

  – An internal CICSRequest node exception is detected.

  – An invalid message body. For example, a parsing error or input message is creating a structure that is larger than the `Commarea length` Basic property that is configured in the CICSRequest node. A similar situation for the returning COMMAREA cannot be detected. For example, if the returning data is larger than the value that is defined in the `Commarea length` property, a memory leak occurs in CICS. Therefore the `Commarea length` Basic property must be correctly configured.

Results

You can use the CICS Connection policy to change connection details for the CICSRequest node (see "Changing connection information for the CICSRequest node" on page 1194).

Changing connection information for the CICSRequest node

You can create a policy that the CICSRequest node or message flow refers to at run time for connection information, instead of defining the connection properties on the node or the message flow. The advantage being that you can change the host name and performance values without needing to redeploy your message flow.

Before you begin

- Read "CICS Transaction Server for z/OS overview" on page 1170 for background information.

About this task

Use the CICS Connection policy to change the CICS Transaction Server for z/OS connection information for the CICSRequest node (see CICS Connection policy (CICSConnection)). The advantage of using a policy is that you can change the host name, performance, and security identity values without needing to redeploy your message flow.

You can use the CICS Connection policy to configure the CICSRequest node to use Secure Sockets Layer (SSL) protocol (see "Securing the connection to CICS Transaction Server for z/OS by using SSL" on page 1182).

Procedure

- Create a CICS Connection policy by using the Policy editor in the IBM App Connect Enterprise Toolkit (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
- Set the properties of the policy to appropriate values, then save the policy.
- Ensure that the CICS server property on the CICSRequest node is set to the name of the policy. The policy name must be in the format *{policyProjectName}:PolicyName*.
- You can deploy the CICS Connection policy in the same BAR file as the associated CICSRequest node, or in a standalone BAR file. If you deploy the policy in a standalone BAR file, you must deploy it before you start the message flow that contains the CICSRequest node.

Propagating security credentials to CICS Transaction Server for z/OS
The CICSRequest node can use an identity that is present in the Properties folder of the message tree structure for the security credentials in a request, by using the `Propagate` property on the security profile that is defined on the node.

About this task

If a CICSRequest node is configured with a security profile, it extracts security tokens from the input message at run time, and propagates an identity to CICS.

Procedure

To propagate an identity to be used for the CICS request security credentials, complete the following steps.

1. Ensure that an appropriate security profile exists for the CICSRequest node, or create a security profile, by following the instructions in "Creating a security profile" on page 2584.
2. Use the BAR editor to select a security profile for the CICSRequest node that has identity propagation enabled.

   For detailed instructions, see "Configuring a message flow for identity propagation" on page 2613.

Working with Salesforce

IBM App Connect Enterprise provides a SalesforceRequest node, which enables you to communicate with Salesforce.com to create, retrieve, update, and delete Salesforce records.

About this task

IBM App Connect Enterprise communicates synchronously with Salesforce by using Loopback connector technology from Strongloop. The Loopback connector invokes Salesforce by using the Force.com REST API to perform create, retrieve, update, and delete operations on predefined and custom Salesforce object types in a Salesforce system. For more information, see the Force.com REST API Developer Guide. The SalesforceRequest node is available on Windows and Linux x64 systems, in all operation modes.

The following topics contain information that you need to understand before you can use the Salesforce support in IBM App Connect Enterprise:

- "Salesforce overview" on page 1196

See the node for information about configuring and using the SalesforceRequest node.

See the following topics for information about the tasks involved in communicating with Salesforce, and in accessing, creating, and modifying Salesforce records:

- "Configuring a secure connection to Salesforce.com" on page 1196
- "Accessing and creating Salesforce data in a message flow " on page 1197
- "Creating Salesforce records" on page 1198
- "Retrieving Salesforce records " on page 1199
- "Filtering the records retrieved from Salesforce" on page 1200
- "Updating Salesforce records" on page 1204
- "Deleting Salesforce records " on page 1205
- "Using local environment variables with Salesforce nodes" on page 1211
- "Using Salesforce models" on page 1212

See "Example: Retrieving all Account records from Salesforce" on page 1202 for a step-by-step example of using a SalesforceRequest node in a message flow.

Salesforce overview

IBM App Connect Enterprise enables you to interact with your Salesforce.com account in the cloud, by using a SalesforceRequest node in a message flow. You can use this capability to create, retrieve, update, and delete records in your Salesforce system directly from IBM App Connect Enterprise.

If you use Salesforce to manage your customer information and interactions, you can now use the SalesforceRequest node in IBM App Connect Enterprise to directly access your Salesforce data and update it with the latest information. This capability provides a way of integrating and simplifying the interactions between the systems in your existing infrastructure.

For information about using the SalesforceRequest node to interact with Salesforce, see "Working with Salesforce" on page 1195.

Configuring a secure connection to Salesforce.com

Access your security information through your Salesforce.com account, and then use this information to configure a security identity by using the **mqsisetdbparms** command in IBM App Connect Enterprise.

Before you begin

Decide which credentials IBM App Connect Enterprise will use when accessing Salesforce.com.

About this task

Follow these steps to create a Salesforce Connected App for use with IBM App Connect Enterprise, and to create a secure connection between the systems:

Procedure

Follow these steps to create a Connected App:

1. Log on to Salesforce.com using your chosen credentials and navigate to the **Apps** page.
2. Create a new Connected App for use with IBM App Connect Enterprise, and set the following properties:
   a) Set the `Connected App Name` property to `IBM Integration Bus`
   b) Set the `API Name` to `IBM_Integration_Bus`
   c) Specify your chosen contact email address.
   d) Select the `Enable OAuth Setting` check box.
   e) Set the `Callback URL` to any valid secure URL (this URL will not be used by your IBM App Connect Enterprise Connected App).
   f) Set the `Selected OAuth scopes` to `Access and manage your data (api).`
   g) Click **Save**.
3. Click **Manage** and ensure that the OAuth policy **Permitted Users** is set to `All users may self-authorize`. If this option is not selected, connections might fail even if the correct credentials are supplied.

Follow these steps to access the required security information from your Salesforce system:

4. Make a note of your Consumer Key and your Consumer Secret, which you will need later when you configure security for the connection.

   The Consumer Key and Consumer secret are generated by Salesforce when you create a Connected App.

5. Obtain the security token for the chosen Salesforce credentials. You can get this from the email that was sent to the email address associated with the credentials when the security token was last reset. Alternatively, you can reset it now by following these steps:
   a) From the Setup page, click your name and then click the **My Settings** menu option.
   b) Click **Reset My Security Token**.
      A message is displayed, informing you that the existing security token will be invalidated.
   c) Click the **Reset Security Token** button to confirm that you want to continue.
      The new security token is sent to your email address.

   For more information, see https://help.salesforce.com/apex/HTViewHelpDoc?id=basics_nav_personal_settings.htm&language=en.

Follow these steps to configure a secure connection between IBM App Connect Enterprise and your Salesforce system:

6. Use the **mqsisetdbparms** command to configure a security identity for the connection, by setting the following parameters:

   -n *salesforce::securityIdentity*
   The name of the security identity that is used to authenticate a connection to a Salesforce system, where *securityIdentity* is the value of the `Security Identity` property in the SalesforceRequest node.

   -u *UserId*
   The Salesforce user ID

   -p *Password*
   The password for accessing the Salesforce system, suffixed with the security token that was sent to you by Salesforce when you reset your security token (in Step 4)

   -c *ClientIdentity*
   The name of the consumer key of your Connected App.

   -s *ClientSecret*
   The consumer secret of your Connected App.

   For example:

   ```
   mqsisetdbparms -w c:\workdir\ACEServ1 -n salesforce::sf1 -u myUserID -p myPassword -
   c myClientIdentity -s myClientSecret
   ```

   For more information, see command. For information about the security credentials that have been set on the integration server, see command.

7. In your message flow, specify `sf1` as the value in the `Security identity` property in the SalesforceRequest node.
   This security identity will be used when you connect to your Salesforce system from IBM App Connect Enterprise.

Accessing and creating Salesforce data in a message flow

You can use the SalesforceRequest node to communicate with Salesforce.com, enabling you to create, retrieve, update, and delete Salesforce records.

Before you begin

Configure a secure connection to Salesforce.com (see “Configuring a secure connection to Salesforce.com” on page 1196).

About this task

The following topics explain how to work with records in a Salesforce system:

- “Creating Salesforce records” on page 1198
- “Retrieving Salesforce records ” on page 1199

- "Updating Salesforce records" on page 1204
- "Deleting Salesforce records " on page 1205

JSON schema are supplied for a wide range of Salesforce objects, and you can use them as input and output for message maps in a Mapping node. For example, you can create the message tree for a Salesforce Account record in a Mapping node by using the JSON schema for the Account object. For more information, see "Using Salesforce models" on page 1212.

Creating Salesforce records
Use the SalesforceRequest node to create a new record in the Salesforce system, for a specified Salesforce object such as an Account.

About this task

You can create a new Salesforce record by specifying `Create` in the `Operation` property of the SalesforceRequest node, and specifying the Salesforce resource type (such as Account) in the `Salesforce object` property. The object can be overridden by the `LocalEnvironment.Destination.Salesforce.Request.object` local environment variable.

The record that you are creating must exist in the input message tree as a JSON object.

When the Salesforce record is created, the input message tree is copied to the location specified by the `Output data location` property, and augmented with the newly assigned Salesforce ID in an *Id* element of the message. If an *Id* element is already in the input message tree, it is ignored.

JSON schema are supplied for a wide range of Salesforce objects, and you can use them as input and output for message maps in a Mapping node. For more information, see "Using Salesforce models" on page 1212.

Procedure

1. Set up a Salesforce account on Salesforce.com.
2. Configure the connection to Salesforce.com.
   See "Configuring a secure connection to Salesforce.com" on page 1196 for more information.
3. Create a flow containing a SalesforceRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Salesforce URL` property, specify the URL of the external Salesforce system.
      - In the `Operation` property, specify `Create`.
      - In the `Salesforce object` property, specify the Salesforce object (for example, `Account`) for which you are creating a new record.
      - In the `Security identity` property, specify the security identity that you will use when connecting to the Salesforce system. For information about configuring the security identity on the integration node, see "Configuring a secure connection to Salesforce.com" on page 1196.
      - In the `Timeout` property, specify the time (in milliseconds) that the node waits for Salesforce to process the operation.
   b) On the **Request** tab, set the `Data location` property to specify the location in the incoming message tree that contains the JSON object data to be created in Salesforce; this data forms the request that is sent from the SalesforceRequest node to the Salesforce system.
   c) On the **Result** tab, set the `Output data location` property to specify the location in the output message tree that will contain the data of the record that is created in Salesforce.

   For more information, see node.
4. Send the required message through the flow to the Salesforce system.
   You will receive a confirmation message from the Salesforce system, which will include the Salesforce ID of the new record.
5. Go to your Salesforce account to verify that the new record has been created correctly.

Retrieving Salesforce records
Use the SalesforceRequest node to retrieve records from the Salesforce system.

About this task

You can retrieve Salesforce records by specifying `Retrieve` in the `Operation` property of the SalesforceRequest node. You can retrieve all records of a particular type by specifying the object type in the `Salesforce object` property of the node. This object type can be overridden by a value specified in the `LocalEnvironment.Destination.Salesforce.Request.object` local environment variable.

Alternatively, you can retrieve a specific record by specifying a Salesforce ID in the `LocalEnvironment.Destination.Salesforce.Request.id` environment variable. The returned record appears as a JSON object in the location that is specified by the `Output data location` property of the node.

If an external ID name and value are specified in the `LocalEnvironment.Destination.Salesforce.Request.externalIdName` and `LocalEnvironment.Destination.Salesforce.Request.externalId` environment variables, all records with that external ID are returned. The returned records appear as a JSON array in the location specified by the `Output data location` property.

If neither a Salesforce ID nor an external ID is specified, all records of the specified type are returned by default.

You can control the records that are returned by using the `LocalEnvironment.Destination.Salesforce.Request.filter` environment variable to specify a filter with one of the following clauses:

- `where`
- `limit`
- `skip`
- `order`
- `field`

You can control which records are returned by specifying a filter with a `where`, `limit`, or `skip` clause. If no ID has been specified, this filter is used to select the records to be returned.

If you specify a filter with an `order` clause, the records are returned in the specified order. For more information, see "Filtering the records retrieved from Salesforce" on page 1200.

You can also restrict the records that are returned by specifying a filter with a `field` clause, so that only the specified field is returned. Filtering with a `field` clause optimizes performance by reducing unnecessary processing.

By default, the retrieval of records is determined by the following properties, in order of precedence:

- Salesforce ID
- External ID
- Filter

If no Salesforce records are found, the output tree contains nothing under JSON.data.

JSON schema are supplied for a wide range of Salesforce objects, and you can use them as input and output for message maps in a Mapping node. For example, you can transform the message tree for a returned Salesforce Account record in a Mapping node by using the JSON schema for the Account object. For more information, see "Using Salesforce models" on page 1212.

Procedure

Follow these steps to retrieve Salesforce records:

1. Set up a Salesforce account on Salesforce.com.

2. Configure a connection to Salesforce.com.

   For more information, see “Configuring a secure connection to Salesforce.com” on page 1196
3. Create a flow containing a SalesforceRequest node, and set the following properties on the **Basic** tab:
   a) In the `Salesforce URL` property, specify the URL of the external Salesforce system.
   b) In the `Operation` property, specify `Retrieve`.
   c) In the `Salesforce object` property, select the type of Salesforce records that you want to retrieve.
   d) In the `Security identity` property, specify the security identity that you will use when connecting to the Salesforce system. For information about configuring the security identity on the integration node, see “Configuring a secure connection to Salesforce.com” on page 1196.
   e) In the `Timeout` property, specify the time (in milliseconds) that the node waits for Salesforce to process the operation.

   For more information, see node.
4. Send the required message through the flow to the Salesforce system.

   You will receive a response from the Salesforce system to confirm that the records have been retrieved.

Example

See “Example: Retrieving all Account records from Salesforce” on page 1202 for a specific example of how to use the SalesforceRequest node to retrieve records from a Salesforce system.

Filtering the records retrieved from Salesforce
Use the **`filter`** element in the LocalEnvironment.Destination.Salesforce.Request.filter environment variable to restrict the records that are retrieved from the Salesforce system.

About this task

You can restrict the records that are returned by using the LocalEnvironment.Destination.Salesforce.Request.filter environment variable to specify a filter with a `where`, `limit`, or `skip` clause. You can also specify a filter with an `order` clause, and the records are returned in the specified order. The returned records appear as a JSON array in the location that is specified by the `Output data location` property of the SalesforceRequest node. By default, records are returned with all fields populated even if the field value is *null*. If a filter is specified with a `field` clause, only the specified field is returned.

The LocalEnvironment.Destination.Salesforce.Request.filter environment variable enables you to specify filter clauses, using child elements of the **`filter`** element:

```
filter
     where <value>
     limit <value>
     skip <value>
     order (1..n)
          <fieldname>/<value>
     field (1..n)
          <fieldname>/<value>
```

You can specify the child elements of the **`filter`** element to restrict the Salesforce records that are retrieved. The available elements are shown in the following table:

Table 38. Elements used for filtering the records retrieved from a Salesforce system		
Element	**Type**	**Description**
where	string	A condition expression used to filter the set of returned records. The expression syntax is a Salesforce Object Query Language (SOQL) condition expression.
limit	positive integer	Imposes a maximum number of records in the returned set.
skip	positive integer	Removes the first *n* records from the returned set.
order	structure	One child, the name of which must match the name of a field of the record. The value is either *ASC* or *DESC* (case-sensitive) and is used to order the returned set. You can specify multiple `order` clauses, which are applied in the order in which they appear in the `filter` tree.
field	structure	One child, the name of which must match the name of a field of the record. The value can be either *true* or *false* (or *1* or *0*), and controls whether the field is included as part of each returned record. You can specify multiple `field` clauses.

For example, to retrieve the Name and Phone fields for 5 (limit) Contact objects in order of Name (ascending), Email (ascending), and Phone (descending), ignoring the first 2 (skip) records, specify the following statements:

```
SET OutputLocalEnvironment.Destination.Salesforce.Request.operation = 'retrieve';
SET OutputLocalEnvironment.Destination.Salesforce.Request.object = 'Contact';
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.limit = 5;
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.skip = 2;
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.order[1].Name = 'ASC';
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.order[2].Email = 'ASC';
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.order[3].Phone = 'DESC';

SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.field[1].Name = true;
SET OutputLocalEnvironment.Destination.Salesforce.Request.filter.field[2].Phone = true;
```

You can also use filtering to retrieve a large number of records in manageable batches, by using repeated invocations of the SalesforceRequest node with different `skip` and `limit` values. For example:

```
/customers?filter[limit]=10&filter[skip]=0
/customers?filter[limit]=10&filter[skip]=10
/customers?filter[limit]=10&filter[skip]=20
```

In this example, each REST request returns ten records; the first request returns the first ten records, the second request returns the next ten records, and the third request returns the next ten records.

Example: Retrieving all Account records from Salesforce
Create a message flow to retrieve all Account records from a Salesforce system by using a SalesforceRequest node.

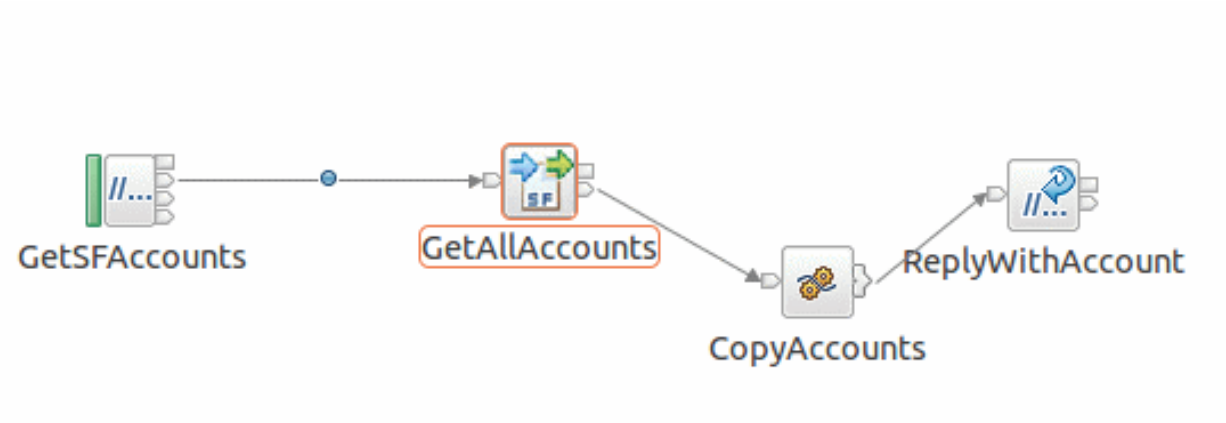
About this task

This message flow receives an XML message over HTTP using an HTTPInput node, and then uses a SalesforceRequest node to send a `Retrieve` request to Salesforce to retrieve all Account records. A Compute node then uses ESQL to move data for each Account into an XML reply message that is returned to the calling client application.

Procedure

Follow these steps to create the required message flow:

1. Create a message flow containing an HTTPInput node (called `GetSFAccounts`), a SalesforceRequest (called `GetAllAccounts`), a Compute node (called `CopyAccounts`), and an HTTPReply node (called `ReplyWithAccount`):



   For information about how to create message flows, see "Creating a message flow" on page 574.
2. On the HTTPInput node, set the following properties:
   a) On the **Description** tab, set the `Node name` property to `GetSFAccounts`
   b) On the **Basic** tab, set the `Path suffix for URL` property to `/sfaccounts`
   c) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.

3. On the SalesforceRequest node, set the following properties:
   a) On the **Description** tab, set the `Node name` property to `GetAllAccounts`
   b) On the **Basic** tab, set the following properties:
      i) Set the `Salesforce URL` property to `https://login.salesforce.com`
      ii) Set the `Operation` property to `Retrieve`
      iii) Set the `Salesforce object` property to `Account`
      iv) Set the `Security identity` property to SF. For information about configuring the security identity, see "Configuring a secure connection to Salesforce.com" on page 1196.
      v) Set the `Timeout (milliseconds)` property to `120000`.
   c) On the **Result** tab, set the following properties:
      i) Set the `Output data location` property to `$OutputLocalEnvironment/SalesforceData`
      ii) Select the `Copy local environment` property.

   The Account records that are returned from the Salesforce system are populated into the output message tree from the SalesforceRequest node under LocalEnvironment.SalesforceData.JSON.Data.
4. On the Compute node, specify the following ESQL statements:

```
CREATE COMPUTE MODULE GetAllSalesForceAccounts_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();

        DECLARE ptrSalesForceAccounts REFERENCE TO
 InputLocalEnvironment.SalesforceData.JSON.Data;

        CREATE FIELD OutputRoot.XMLNSC.SalesforceAccounts;
        DECLARE ptrNewAccount REFERENCE TO OutputRoot.XMLNSC.SalesforceAccounts;

        MOVE ptrSalesForceAccounts FIRSTCHILD TYPE Name NAME 'Item';

        WHILE lastmove(ptrSalesForceAccounts) DO
            -- An Account was returned from Salesforce so populate a child element
            -- called Account as the last child of OutputRoot.XMLNSC.SalesforceAccounts with
 the
            -- Name, Type, BillingStreet and BillingCity of the returned account data
            CREATE LASTCHILD OF ptrNewAccount NAME 'Account';
            MOVE ptrNewAccount LASTCHILD;
            CREATE LASTCHILD OF ptrNewAccount NAME 'Name' VALUE ptrSalesForceAccounts.Name;
            CREATE LASTCHILD OF ptrNewAccount NAME 'Type' VALUE ptrSalesForceAccounts.Type;
            CREATE LASTCHILD OF ptrNewAccount NAME 'BillingStreet' VALUE
 ptrSalesForceAccounts.BillingStreet;
            CREATE LASTCHILD OF ptrNewAccount NAME 'BillingCity' VALUE
 ptrSalesForceAccounts.BillingCity;
            MOVE ptrSalesForceAccounts NEXTSIBLING REPEAT TYPE NAME;
            MOVE ptrNewAccount PARENT;
        END WHILE;

        RETURN TRUE;
    END;

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

```
END MODULE;
```

The Compute node moves the Name, Type, BillingStreet, and BillingCity fields for each Account into an XML reply message, which is returned to the calling client application by using the ESQL statements defined on the node.

The format of the XML message that is returned is shown in the following example:

```
<SalesforceAccounts>
     <Account>
          <Name>....</Name>
          <Type>....</Type>
          <BillingStreet>....</BillingStreet>
          <BillingCity> … .</BillingCity>
     </Account>
     <Account>
          <Name>....</Name>
          <Type>....</Type>
          <BillingStreet>....</BillingStreet>
          <BillingCity> … .</BillingCity>
     </Account>
</SalesforceAccounts>
```

For more information about specifying the ESQL statements, see node.

5. On the HTTPReply node, set the Node name property (on the **Description** tab) to ReplyWithAccount.

Updating Salesforce records
Use the SalesforceRequest node to update records in the Salesforce system.

About this task

You can update Salesforce records by specifying Update in the Operation property of the SalesforceRequest node, and specifying the object type in the Salesforce object property. The object type can be overridden by the LocalEnvironment.Destination.Salesforce.Request.object environment variable.

The fields of the Salesforce record to be updated must appear in the input message tree as a JSON object. Fields that do not require updating do not have to appear in the tree.

If a Salesforce ID is specified in the LocalEnvironment.Destination.Salesforce.Request.Id environment variable, the record with that Salesforce ID value is updated.

If a Salesforce ID is not specified, and an external ID name and value are specified in the LocalEnvironment.Destination.Salesforce.Request.externalIdName and LocalEnvironment.Destination.Salesforce.Request.externalId environment variables, the record with that external ID is updated. If the record does not exist, it is created. If neither a Salesforce ID nor an external ID is specified in the local environment, but an Id element is found in the input message tree, that value is used as a Salesforce ID and the record with that Salesforce ID is updated.

When the record is updated, the input message tree is copied to the location specified by the Output data location property of the node, and augmented with the Salesforce ID in an Id element. The Salesforce Id is not provided in the Output tree if an external Id has been specified, unless a new Salesforce record is created (if it did not previously exist). If an external Id is specified but the Salesforce record cannot be found, the Salesforce record is created. This is called an *upsert* operation and occurs only if an external Id has been specified.

JSON schema are supplied for a wide range of Salesforce objects, and you can use them as input and output for message maps in a Mapping node. For example, you can create the message tree to update a Salesforce Account record in a Mapping node by using the JSON schema for the Account object. For more information, see “Using Salesforce models” on page 1212.

Procedure

Follow these steps to update a Salesforce record of a specified type (for example, an Account object):

1. Set up a Salesforce account on Salesforce.com.
2. Configure a connection to Salesforce.com.
   For more information, see "Configuring a secure connection to Salesforce.com" on page 1196.
3. Create a flow containing a SalesforceRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Salesforce URL` property, specify the URL of the external Salesforce system.
      - In the `Operation` property, specify `Update`.
      - In the `Salesforce object` property, select the type of Salesforce object that you want to update (for example, `Account`).
      - In the `Security identity` property, specify the security identity that you will use when connecting to the Salesforce system. For information about configuring the security identity on the integration node, see "Configuring a secure connection to Salesforce.com" on page 1196.
      - In the `Timeout` property, specify the time (in milliseconds) that the node waits for Salesforce to process the operation.
   b) On the **Request** tab, set the `Data location` property to specify the location in the incoming message tree that contains the JSON objects data that will be used to update Salesforce.

   For more information, see node.
4. Send the required message through the flow to the Salesforce system.
   You will receive a response from the Salesforce system to confirm that the records have been updated.
5. Go to your Salesforce account to verify that the records have been updated correctly.

Deleting Salesforce records
Use the SalesforceRequest node to delete a record from a Salesforce system.

About this task

You can delete a Salesforce record by specifying `Delete` in the `Operation` property of the SalesforceRequest node, and specifying the Salesforce ID of the record in the LocalEnvironment.Destination.Salesforce.Request.id environment variable.

When the Salesforce record is deleted, a JSON object containing the Salesforce ID of the deleted record appears in the location that is specified by the `Output data location` property of the node.

Procedure

Follow these steps to delete a Salesforce record:

1. Set up a Salesforce account on Salesforce.com.
2. Configure a connection to Salesforce.com.
   For more information, see "Configuring a secure connection to Salesforce.com" on page 1196.

3. Create a flow containing a SalesforceRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Salesforce URL` property, specify the URL of the external Salesforce system.
      - In the `Operation` property, specify `Delete`.
      - In the `Salesforce object` property, select the type of Salesforce object that you want to delete.
      - In the `Security identity` property, specify the security identity that you will use when connecting to the Salesforce system. For information about configuring the security identity on the integration node, see "Configuring a secure connection to Salesforce.com" on page 1196.
      - In the `Timeout` property, specify the time (in milliseconds) that the node waits for Salesforce to process the operation.
   b) On the **Results** tab, set the `Output data location` property to control where the JSON object that contains the ID of the deleted record will be placed in the output message tree.

   For more information, see node.
4. Ensure that the Salesforce ID of the record that you want to delete is set in LocalEnvironment.Destination.Salesforce.Request.id.
5. Send the required message through the flow to the Salesforce system.

   If the delete operation is successful, the output tree will contain the ID of the deleted Salesforce record.
6. Go to your Salesforce account to verify that the record has been deleted successfully.

SalesforceRequest node: Input and output message trees

Following a `create` or `update` operation, the output message body contains the input message body augmented with data from the Salesforce system. Following a `retrieve` or `delete` operation, the input message body is discarded and replaced.

The data that is added varies according to the operation performed by the SalesforceRequest node (create, retrieve, update, or delete):

- Following a `create` operation, the output message body contains the input message body augmented with the Salesforce ID of the newly created Salesforce record
- Following a `retrieve` operation, the output message body contains the following data:
  - If you specified a Salesforce ID, the output message body contains a JSON object for the record
  - If you did not specify a Salesforce ID, the output message body contains a JSON array for zero or more records
- Following an `update` operation, the output message body contains the following data:
  - If you specified a Salesforce ID, the output message body contains the input message body augmented with the Salesforce ID of the updated Salesforce record
  - If you specified an External ID and the Salesforce record already existed, the output message body contains the input message body
  - If you specified an External ID and the Salesforce record did not previously exist, the output message body contains the input message body augmented with the Salesforce ID of the newly created Salesforce record
- Following a `delete` operation, the output message body contains the Salesforce ID of the Salesforce record that was deleted.

The following table shows example input and output message bodies for each operation:

Table 39. SalesforceRequest node - input and output message bodies. Example input message bodies to the SalesforceRequest node and the output message bodies following a create, retrieve, update, or delete operation.

Operation	input message body to SalesforceRequest node	output message body from SalesforceRequest node
create	(0x01000000:Object):JSON = (['json' : 0x7fe8bc1b65c0] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-32' (CHARACTER) (0x03000000:NameValue):Type = 'good' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'Romsey Road' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Winchester' (CHARACTER) (0x03000000:NameValue):BillingState = 'Hampshire' (CHARACTER) (0x03000000:NameValue):BillingPostalCode = 'SO21 2JN' (CHARACTER) (0x03000000:NameValue):BillingCountry = 'United Kingdom' (CHARACTER) (0x03000000:NameValue):Phone = '01962815000' (CHARACTER) (0x03000000:NameValue):Fax = '01962816666' (CHARACTER) (0x03000000:NameValue):AccountNumber = '001122334455' (CHARACTER) (0x03000000:NameValue):Website = 'http://www.ibm.com' (CHARACTER)))	(0x01000000:Object):JSON = (['json' : 0x7fe8dc2d9a70] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-32' (CHARACTER) (0x03000000:NameValue):Type = 'good' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'Romsey Road' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Winchester' (CHARACTER) (0x03000000:NameValue):BillingState = 'Hampshire' (CHARACTER) (0x03000000:NameValue):BillingPostalCode = 'SO21 2JN' (CHARACTER) (0x03000000:NameValue):BillingCountry = 'United Kingdom' (CHARACTER) (0x03000000:NameValue):Phone = '01962815000' (CHARACTER) (0x03000000:NameValue):Fax = '01962816666' (CHARACTER) (0x03000000:NameValue):AccountNumber = '001122334455' (CHARACTER) (0x03000000:NameValue):Website = 'http://www.ibm.com' (CHARACTER) (0x03000000:NameValue):Id = '00158000003M6LzAAK' (CHARACTER)))

Table 39. SalesforceRequest node - input and output message bodies. Example input message bodies to the SalesforceRequest node and the output message bodies following a create, retrieve, update, or delete operation. *(continued)*

Operation	input message body to SalesforceRequest node	output message body from SalesforceRequest node
retrieve (by Id)	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x03000000:NameValue):id = '00158000003M6LzAAK' (CHARACTER)))) No Input data	(0x01000000:Object):JSON = (['json' : 0x7fe8dc376bd0] (0x01000000:Object):Data = ((0x03000000:NameValue):Id = '00158000003M6LzAAK' (CHARACTER) (0x01000000:Object):attributes = ((0x03000000:NameValue):type = 'Account' (CHARACTER) (0x03000000:NameValue):url = '/services/data/v34.0/sobjects/Account/00158000003M6LzAAK' (CHARACTER)) (0x03000000:NameValue):IsDeleted = FALSE (BOOLEAN) (0x03000000:NameValue):MasterRecordId = NULL (0x03000000:NameValue):Name = 'Account-32' (CHARACTER) (0x03000000:NameValue):Type = 'good' (CHARACTER) (0x03000000:NameValue):ParentId = NULL (0x03000000:NameValue):BillingStreet = 'Romsey Road' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Winchester' (CHARACTER) (0x03000000:NameValue):BillingState = 'Hampshire' (CHARACTER) (0x03000000:NameValue):BillingPostalCode = 'SO21 2JN' (CHARACTER) (0x03000000:NameValue):BillingCountry = 'United Kingdom' (CHARACTER) (0x03000000:NameValue):BillingLatitude = NULL (0x03000000:NameValue):BillingLongitude = NULL (0x01000000:Object):BillingAddress = ((0x03000000:NameValue):city = 'Winchester' (CHARACTER) (0x03000000:NameValue):country = 'United Kingdom' (CHARACTER) (0x03000000:NameValue):countryCode = NULL (0x03000000:NameValue):geocodeAccuracy = NULL (0x03000000:NameValue):latitude = NULL (0x03000000:NameValue):longitude = NULL (0x03000000:NameValue):postalCode = 'SO21 2JN' (CHARACTER) (0x03000000:NameValue):state = 'Hampshire' (CHARACTER) (0x03000000:NameValue):stateCode = NULL (0x03000000:NameValue):street = 'Romsey Road' (CHARACTER)) (0x03000000:NameValue):ShippingStreet = NULL (0x03000000:NameValue):ShippingCity = NULL (0x03000000:NameValue):ShippingState = NULL (0x03000000:NameValue):ShippingPostalCode = NULL (0x03000000:NameValue):ShippingCountry = NULL (0x03000000:NameValue):ShippingLatitude = NULL (0x03000000:NameValue):ShippingLongitude = NULL (0x03000000:NameValue):ShippingAddress = NULL (0x03000000:NameValue):Phone = '01962815000' (CHARACTER) (0x03000000:NameValue):Fax = '01962816666' (CHARACTER) (0x03000000:NameValue):AccountNumber = '001122334455' (CHARACTER) (0x03000000:NameValue):Website = 'http://www.ibm.com' (CHARACTER) (0x03000000:NameValue):PhotoUrl = '/services/images/photo/00158000003M6LzAAK' (CHARACTER) (0x03000000:NameValue):Sic = NULL (0x03000000:NameValue):Industry = NULL (0x03000000:NameValue):AnnualRevenue = NULL (0x03000000:NameValue):NumberOfEmployees = NULL (0x03000000:NameValue):Ownership = NULL (0x03000000:NameValue):TickerSymbol = NULL (0x03000000:NameValue):Description = NULL (0x03000000:NameValue):Rating = NULL (0x03000000:NameValue):Site = NULL (0x03000000:NameValue):OwnerId = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):CreatedDate = '2016-02-17T13:39:30.000+0000' (CHARACTER) (0x03000000:NameValue):CreatedById = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):LastModifiedDate = '2016-02-17T13:39:30.000+0000' (CHARACTER) (0x03000000:NameValue):LastModifiedById = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):SystemModstamp = '2016-02-17T13:39:30.000+0000' (CHARACTER) (0x03000000:NameValue):LastActivityDate = NULL (0x03000000:NameValue):LastViewedDate = '2016-02-17T13:39:30.000+0000' (CHARACTER) (0x03000000:NameValue):LastReferencedDate = '2016-02-17T13:39:30.000+0000' (CHARACTER) (0x03000000:NameValue):Jigsaw = NULL (0x03000000:NameValue):JigsawCompanyId = NULL (0x03000000:NameValue):CleanStatus = 'Pending' (CHARACTER) (0x03000000:NameValue):AccountSource = NULL (0x03000000:NameValue):DunsNumber = NULL (0x03000000:NameValue):Tradestyle = NULL (0x03000000:NameValue):NaicsCode = NULL (0x03000000:NameValue):NaicsDesc = NULL (0x03000000:NameValue):YearStarted = NULL (0x03000000:NameValue):SicDesc = NULL (0x03000000:NameValue):DandbCompanyId = NULL (0x03000000:NameValue):CustomerPriority__c = NULL (0x03000000:NameValue):SLA__c = NULL (0x03000000:NameValue):Active__c = NULL (0x03000000:NameValue):NumberofLocations__c = NULL (0x03000000:NameValue):UpsellOpportunity__c = NULL (0x03000000:NameValue):SLASerialNumber__c = NULL (0x03000000:NameValue):SLAExpirationDate__c = NULL

Table 39. SalesforceRequest node - input and output message bodies. Example input message bodies to the SalesforceRequest node and the output message bodies following a create, retrieve, update, or delete operation. *(continued)*

Operation	input message body to SalesforceRequest node	output message body from SalesforceRequest node
retrieve using a filter (filter.limit =2)	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x01000000:Name):filter = ((0x03000000:NameValue):limit = 2 (INTEGER)))))	(0x01000000:Object):JSON = (['json' : 0x7f80d02f0710] (0x01001000:Array):Data = ((0x01000000:Object):Item = ((0x03000000:NameValue):Id = '00158000002dQ2eAAE' (CHARACTER) (0x01000000:Object):attributes = ((0x03000000:NameValue):type = 'Account' (CHARACTER) (0x03000000:NameValue):url = '/services/data/v34.0/sobjects/Account/00158000002dQ2eAAE' (CHARACTER)) (0x03000000:NameValue):IsDeleted = FALSE (BOOLEAN) (0x03000000:NameValue):MasterRecordId = NULL (0x03000000:NameValue):Name = 'One' (CHARACTER) (0x03000000:NameValue):Type = NULL (0x03000000:NameValue):ParentId = NULL (0x03000000:NameValue):BillingStreet = 'Sanjay Billing Street' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Southampton' (CHARACTER) (0x03000000:NameValue):BillingState = NULL (0x03000000:NameValue):BillingPostalCode = NULL (0x03000000:NameValue):BillingCountry = NULL (0x03000000:NameValue):BillingLatitude = NULL (0x03000000:NameValue):BillingLongitude = NULL (0x01000000:Object):BillingAddress = ((0x03000000:NameValue):city = 'Southampton' (CHARACTER) (0x03000000:NameValue):country = NULL (0x03000000:NameValue):countryCode = NULL (0x03000000:NameValue):geocodeAccuracy = NULL (0x03000000:NameValue):latitude = NULL (0x03000000:NameValue):longitude = NULL (0x03000000:NameValue):postalCode = NULL (0x03000000:NameValue):state = NULL (0x03000000:NameValue):stateCode = NULL (0x03000000:NameValue):street = 'Sanjay Billing Street' (CHARACTER)) (0x03000000:NameValue):ShippingStreet = 'Sanjay Shipping Street' (CHARACTER) (0x03000000:NameValue):ShippingCity = 'Southampton' (CHARACTER) (0x03000000:NameValue):ShippingState = NULL (0x03000000:NameValue):ShippingPostalCode = NULL (0x03000000:NameValue):ShippingCountry = NULL (0x03000000:NameValue):ShippingLatitude = NULL (0x03000000:NameValue):ShippingLongitude = NULL (0x01000000:Object):ShippingAddress = ((0x03000000:NameValue):city = 'Southampton' (CHARACTER) (0x03000000:NameValue):country = NULL (0x03000000:NameValue):countryCode = NULL (0x03000000:NameValue):geocodeAccuracy = NULL (0x03000000:NameValue):latitude = NULL (0x03000000:NameValue):longitude = NULL (0x03000000:NameValue):postalCode = NULL (0x03000000:NameValue):state = NULL (0x03000000:NameValue):stateCode = NULL (0x03000000:NameValue):street = 'Sanjay Shipping Street' (CHARACTER)) (0x03000000:NameValue):Phone = NULL (0x03000000:NameValue):Fax = NULL (0x03000000:NameValue):AccountNumber = '12341234' (CHARACTER) (0x03000000:NameValue):Website = NULL (0x03000000:NameValue):PhotoUrl = '/services/images/photo/00158000002dQ2eAAE' (CHARACTER) (0x03000000:NameValue):Sic = NULL (0x03000000:NameValue):Industry = NULL (0x03000000:NameValue):AnnualRevenue = NULL (0x03000000:NameValue):NumberOfEmployees = NULL (0x03000000:NameValue):Ownership = NULL (0x03000000:NameValue):TickerSymbol = NULL (0x03000000:NameValue):Description = NULL (0x03000000:NameValue):Rating = NULL (0x03000000:NameValue):Site = NULL (0x03000000:NameValue):OwnerId = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):CreatedDate = '2016-01-13T22:26:03.000+0000' (CHARACTER) (0x03000000:NameValue):CreatedById = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):LastModifiedDate = '2016-01-19T12:35:55.000+0000' (CHARACTER) (0x03000000:NameValue):LastModifiedById = '00558000000a0qdAAA' (CHARACTER) (0x03000000:NameValue):SystemModstamp = '2016-01-19T12:35:55.000+0000' (CHARACTER) (0x03000000:NameValue):LastActivityDate = NULL (0x03000000:NameValue):LastViewedDate = '2016-01-19T12:35:55.000+0000' (CHARACTER) (0x03000000:NameValue):LastReferencedDate = '2016-01-19T12:35:55.000+0000' (CHARACTER) (0x03000000:NameValue):Jigsaw = NULL (0x03000000:NameValue):JigsawCompanyId = NULL (0x03000000:NameValue):CleanStatus = 'Pending' (CHARACTER) (0x03000000:NameValue):AccountSource = NULL (0x03000000:NameValue):DunsNumber = NULL (0x03000000:NameValue):Tradestyle = NULL (0x03000000:NameValue):NaicsCode = NULL (0x03000000:NameValue):NaicsDesc = NULL (0x03000000:NameValue):YearStarted = NULL (0x03000000:NameValue):SicDesc = NULL (0x03000000:NameValue):DandbCompanyId = NULL

Table 39. SalesforceRequest node - input and output message bodies. Example input message bodies to the SalesforceRequest node and the output message bodies following a create, retrieve, update, or delete operation. *(continued)*

Operation	input message body to SalesforceRequest node	output message body from SalesforceRequest node
update (using Id)	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x03000000:NameValue):Id = '00158000003M6LzAAK' (CHARACTER)))) (0x01000000:Object):JSON = (['json' : 0x7fe88c359150] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-32' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'Test Road' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Hedge End' (CHARACTER) (0x03000000:NameValue):BillingState = 'Hampshire' (CHARACTER) (0x03000000:NameValue):BillingPostalCode = 'SO30' (CHARACTER) (0x03000000:NameValue):BillingCountry = 'United Kingdom' (CHARACTER)))	(0x01000000:Object):JSON = (['json' : 0x7fe8dc43c460] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-32' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'Test Road' (CHARACTER) (0x03000000:NameValue):BillingCity = 'Hedge End' (CHARACTER) (0x03000000:NameValue):BillingState = 'Hampshire' (CHARACTER) (0x03000000:NameValue):BillingPostalCode = 'SO30' (CHARACTER) (0x03000000:NameValue):BillingCountry = 'United Kingdom' (CHARACTER) (0x03000000:NameValue):Id = '00158000003M6LzAAK' (CHARACTER)))
update (using external Id and the Salesforce record did not already exist)	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x03000000:NameValue):externalIdName = 'MyExternalId__c' (CHARACTER) (0x03000000:NameValue):externalId = 'special4' (CHARACTER)))) (0x01000000:Object):JSON = (['json' : 0x7f80f817c0a0] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-82' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'New Road' (CHARACTER)))	(0x01000000:Object):JSON = (['json' : 0x7f80d0188760] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-82' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'New Road' (CHARACTER) (0x03000000:NameValue):Id = '00158000003MBiuAAG' (CHARACTER)))
update (using external Id and the Salesforce record already existed)	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x03000000:NameValue):externalIdName = 'MyExternalId__c' (CHARACTER) (0x03000000:NameValue):externalId = 'special4' (CHARACTER)))) (0x01000000:Object):JSON = (['json' : 0x7f80f817c0a0] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-82' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'New Road' (CHARACTER)))	(0x01000000:Object):JSON = (['json' : 0x7f80d0188760] (0x01000000:Object):Data = ((0x03000000:NameValue):Name = 'Account-82' (CHARACTER) (0x03000000:NameValue):BillingStreet = 'New Road' (CHARACTER)))
delete	LocalEnvironment: (0x01000000:Name):Destination = ((0x01000000:Name):Salesforce = ((0x01000000:Name):Request = ((0x03000000:NameValue):id = '00158000003M82WAAS' (CHARACTER)))) No Input data	(0x01000000:Object):JSON = (['json' : 0x7fe128478650] (0x01000000:Object):Data = ((0x03000000:NameValue):Id = '00158000003M82WAAS' (CHARACTER)))

Using local environment variables with Salesforce nodes

The SalesforceRequest node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node properties.

The following table shows the environment variables that can be set to control the behavior of the SalesforceRequest node:

Local environment variable	Type	Description
LocalEnvironment.Destination.Salesforce.RequestURL	String	The URL of the external Salesforce system. Overrides the **Salesforce URL** property on the SalesforceRequest node.
LocalEnvironment.Destination.Salesforce.RequestOperation	String	The operation to be requested on the Salesforce system: • `Create` • `Update` • `Retrieve` • `Delete` This environment variable overrides the **Operation** property on the SalesforceRequest node.
LocalEnvironment.Destination.Salesforce.RequestObject	String	The Salesforce object type for the request. This environment variable overrides the **Salesforce object** property on the SalesforceRequest node.
LocalEnvironment.Destination.Salesforce.RequestId	String	The Salesforce ID of a Salesforce record. This value is used for the following operations: • `Retrieve` (optional) • `Update` (optional) • `Delete` (required) This Salesforce `Id` is assigned to a record when it is created on the Salesforce system.
LocalEnvironment.Destination.Salesforce.RequestExternalIdName	String	The name of an external ID field of a Salesforce record. This value is used for the following operations: • `Retrieve` (optional) • `Update` (optional)
LocalEnvironment.Destination.Salesforce.RequestExternalId	String	The external ID of a Salesforce record. This value is used for the following operations: • `Retrieve` (optional) • `Update` (optional)
LocalEnvironment.Destination.Salesforce.RequestFilter	String or structure	The filter to be used for restricting the records returned by a `Retrieve` operation. For more information, see "Retrieving Salesforce records " on page 1199.

Local environment variable	Type	Description
LocalEnvironment.Destination.Salesforce.RequestTimeoutMilliseconds	integer	The time (in milliseconds) that the node waits for Salesforce to process the operation. This environment variable overrides the **Timeout (milliseconds)** property on the SalesforceRequest node.

Using Salesforce models

JSON schema are supplied for a wide range of Salesforce objects, and you can use them as input and output for message maps in a Mapping node. For example, you can create the message tree for a new Account record, or transform the message tree for a returned Account record, by using the JSON schema for the Salesforce Account object.

About this task

The JSON schema for Salesforce objects are provided with IBM App Connect Enterprise in the *install_dir*/server/sample/Salesforce directory, where *install_dir* is the directory in which you installed IBM App Connect Enterprise.

Procedure

To reference the Salesforce object schema from a Mapping node, complete the following steps:

1. Create a shared library.

   Note: The map and the JSON schema that it references must all be held in the same shared library.
2. Import into the shared library the schema for the objects that you want to map.
3. If you have added custom fields to your Salesforce objects, edit the schema to add the custom fields, and save it.

   For information about the JSON schema, see JSON schema requirements for message maps.
4. Add a reference to the shared library from the application that contains your message flow.
5. Drag and drop a Mapping node onto the message flow canvas and then double-click the Mapping node to launch the **New Message Map** wizard.
6. Specify a name for the map, change the container to be the shared library that holds the JSON schema, and click **Next**.
7. Select as input or output the required Salesforce object from the list under **JSON schema**. Each object appears as both a JSON object and a JSON array:
   - Use the JSON object if creating a message tree prior to a Salesforce Create or Update operation.
   - Use the JSON object if transforming a message tree after a Salesforce Retrieve operation that used a Salesforce ID.
   - Use the JSON array if transforming a message tree after all other Salesforce Retrieve operations.
8. Select the other output or input to the map and click **Finish**.

Working with LoopBack connectors

IBM App Connect Enterprise provides a LoopBackRequest node, which enables you to issue synchronous requests to backend data sources by using LoopBack connectors such as MongoDB, Cloudant, or PostgreSQL.

About this task

You can use a LoopBackRequest node in a message flow to access or create records through a LoopBack connector. First, you must install your chosen connector to work with IBM App Connect Enterprise, and

then configure the data source for the connector. You can then specify the security credentials that are required to access it. The following topics describe the steps involved in completing those tasks:

1. “Installing the LoopBack connector” on page 1213
2. “Configuring the data source and models for your LoopBack connector” on page 1214
3. “Specifying security credentials for connecting to a secured data source” on page 1216

When you have completed these tasks, you can use a LoopBackRequest node in a message flow to create, retrieve, update, and delete records, as described in the following topics:

- “Accessing and creating records by using the LoopBackRequest node” on page 1217
- “Creating records by using a LoopBackRequest node” on page 1218
- “Retrieving records by using a LoopBackRequest node” on page 1219
- “Filtering the records retrieved through a LoopBack connector” on page 1220
- “Updating records by using a LoopBackRequest node” on page 1223
- “Deleting records by using a LoopBackRequest node” on page 1224

The LoopBackRequest node operations are synchronous and non-transactional, which means that, if a message flow fails and rolls back after the LoopBackRequest node, the operation on the data source will still complete.

The LoopBackRequest node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node properties. For information about defining these environment variables, see “Using local environment variables with LoopBackRequest nodes” on page 1226.

For more information about configuring and using the LoopBackRequest node in a message flow, see node.

When you pass data to a data source through a LoopBackRequest node (when you create or update a record, for example), you must use the JSON domain for providing data in the input message. Replies to a LoopBack request are provided as JSON domain messages.

If you want to use a Mapping node to create the input or process the output data, you must use one of the supported options for defining a message model for the LoopBack connector JSON data. For more information, see Creating or transforming a JSON output message by using a message map.

For help with diagnostics, check the user trace and the LoopBack activity log, which contains information that is created when a LoopBackRequest node issues a request. For information about the type of information that is included, see “LoopBack Activity Log” on page 2865.

Installing the LoopBack connector

Install and configure the required LoopBack connector for use with IBM App Connect Enterprise.

About this task

Before you can use the LoopBackRequest node in a message flow, you must install the relevant LoopBack connector.

Procedure

Follow these steps to install and configure your LoopBack connector to work with IBM App Connect Enterprise:

1. Open the IBM App Connect Enterprise Command Console and change to the `node_modules` subdirectory of the work path directory (which is defined by the MQSI_WORKPATH environment variable).
   For example, on Windows:

   ```
   cd %MQSI_WORKPATH%\node_modules
   ```

   and on UNIX:

   ```
   cd $MQSI_WORKPATH/node_modules
   ```

2. From the `node_modules` subdirectory, run the `npm` command, specifying the name of your chosen LoopBack connector.

   For example, to install the MongoDB LoopBack connector:

   ```
   npm install loopback-connector-mongodb
   ```

   Note: Ensure that you use the version of `npm` that is provided with IBM App Connect Enterprise, by running the `npm` command from the Command Console. If you install the connector by using any other method, the modules will be incompatible.

   Some LoopBack connectors might depend on native libraries that are built during the installation process, and which require compiler tools to be accessible on the system. For example, the LoopBack Oracle connector requires changes to the library path so that the connector can locate the Oracle InstantClient that is packaged with the connector. On Windows, you could enable this by adding the following line to a common profile in the *work_path*`\common\profiles\loopback.cmd` file:

   ```
   set PATH=%PATH%;%MQSI_WORKPATH%\node_modules\instantclient\vc11;%MQSI_WORKPATH%\node_modules
   \instantclient;
   ```

   On Linux, you could enable the Oracle connector to locate the InstantClient, by adding the following line to a common profile in the *work_path*`/common/profiles/loopback.sh` file:

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/$MQSI_WORKPATH/node_modules/instantclient:
   ```

   If you encounter problems when installing LoopBack connectors, contact the connector author for assistance.

   Some LoopBack connectors might also require you to configure the environment in which the connector is running, such as modifying the library path. See “Setting up a command environment” on page 64 for information on how to configure the command environment to make these settings available to IBM App Connect Enterprise.

   Some LoopBack connectors might be incompatible if they use native libraries that are used internally by IBM App Connect Enterprise.
3. Check the `node_modules` subdirectory to confirm that the required LoopBack connector and any dependent modules have been installed.

What to do next

Configure the data source for your installed connector, as described in “Configuring the data source and models for your LoopBack connector” on page 1214.

Configuring the data source and models for your LoopBack connector

Ensure that the data source for your LoopBack connector has been configured.

Before you begin

Install the required LoopBack connector as described in “Installing the LoopBack connector” on page 1213.

About this task

When you have installed a LoopBack connector to be used with IBM App Connect Enterprise, you must configure the data source for the connector, by adding a connector-specific data-source stanza to the `datasources.json` file.

Procedure

Follow these steps to configure the data source for your chosen LoopBack connector:

1. Create or update a LoopBack `datasources.json` file, in the IBM App Connect Enterprise *workpath*`/connectors/loopback` directory. This `datasources.json` file can be specific to the integration node or the integration server.

   In most circumstances, the `/connectors/loopback` directory is created automatically; however, if it does not exist, you must create it.

   The LoopBackRequest node searches for its configured data-source stanza in the following order:

   a. *workpath*/*integration_node*/*integration_server*/connectors/loopback

   b. *workpath*/*integration_node*/connectors/loopback

   c. *workpath*/connectors/loopback

2. Modify the `datasources.json` file by adding a JSON data-source stanza that configures the data source that you want to access.

   The LoopBack `datasources.json` file must provide all the properties that are required by the chosen connector. The details of the properties are defined by the installed LoopBack connector. For connecting to a secured connector, the security credentials can be provided in the `datasources.json` file; however, they are stored in plain text in the `workpath` folder. For an alternative approach to providing security credentials, see "Specifying security credentials for connecting to a secured data source" on page 1216.

   The LoopBack `datasources.json` file can contain properties for multiple data sources; for example, it could contain the following code, which provides data-source stanzas for both a `mongodb_dev` and a `mongodb_stage` MongoDB connector:

   ```
   {
     "mongodb_dev": {
       "name": "mongodb_dev",
       "connector": "mongodb",
       "host": "127.0.0.1",
       "port": 27017,
       "database": "devDB",
       "username": "devUser",
       "password": "devPassword"
     },
     "mongodb_stage": {
       "name": "mongodb_stage",
       "connector": "mongodb",
       "port": 27018,
       "host": "staging.org.co.uk",
       "database": "stageDB"
     }
   }
   ```

   The content of the LoopBack `datasources.json` properties for the data sources are defined and used by the installed LoopBack connector; the LoopBackRequest node passes the JSON stanza to the installed connector when it is initialized at flow deployment or restart.

3. Optional: You can provide a model definition file that is specific to the data-source stanza.

   Some LoopBack connectors require that the data model to be interacted with is defined through a LoopBack JSON model. When the LoopBackRequest node interacts for the first time with a named LoopBack object, which is either specified as a node property or dynamically overridden in the local environment, it looks for a LoopBack JSON model file in a subdirectory of the directory that contains the `datasources.json` file. This subdirectory, which you must create, must have exactly the

same name and case as the data-source stanza that is specified on the LoopBackRequest node. For example, if the `datasources.json` file is provided only at the node location, the model would be:

```
workpath/connectors/loopback/datasource_name/model_name.json
```

The *model_name* part of the *model_name*`.json` file must match exactly (including the case) the **LoopBack object** name that is either specified on the LoopBackRequest node or overridden with the **LocalEnvironment.Destination.Loopback.Request.object** local environment variable.

The name that is specified in the **LoopBack object** property of the LoopBackRequest node is the base name of the model file (excluding the `.json` file extension), and the **name** property in the model file defines the LoopBack object that will be interacted with. Typically, these names are the same, but they can be different; for example, if the name of the object in the LoopBack connected system contains a character that would be invalid in a file name. If the names are different, a BIP3880 message is issued.

The following example shows how you could provide LoopBack JSON model files for working with the `dev` and `staging` MongoDB data sources:

```
<workpath>
   \---connectors
         \---loopback
             +---datasources.json
             |
             +---mongodb_dev
             |   +---customer.json
             |   +---order.json
             |   \---test.json
             |
             \---mongodb_stage
                 +---customer.json
                 \---order.json
```

For more information about using models, see "Using models with LoopBack connectors" on page 1229.

What to do next

Optionally, you can provide security credentials for the LoopBack connector by running the command, as described in "Specifying security credentials for connecting to a secured data source" on page 1216.

Specifying security credentials for connecting to a secured data source

You can use the **mqsisetdbparms** command to specify the security credentials used by a LoopBackRequest node to connect to a secured data source.

About this task

You can associate a user name, password, client identity, and client secret with connections to a secured LoopBack connector. Alternatively, you can specify the security credentials in the `datasources.json` file; however, the data is stored in the file in plain text.

Procedure

Follow these steps to configure the security credentials for connections to a LoopBack connector:

1. Run the **mqsisetdbparms** command, specifying the following parameters:

   -n loopback::*securityIdentity*
   The name of the security identity that is used to authenticate a connection to a LoopBack connector, where *securityIdentity* is the value of the **Security identity** property in the LoopBackRequest node.

   -u *userID*
   The user ID for connecting to the LoopBack connector application.

   -p *password*
   The password for accessing the LoopBack connector application.

   -c *clientIdentity*
   (Optional) The name of the client ID of the LoopBack connector application.

   -s *clientSecret*
   (Optional) The client secret of the LoopBack connector application.

   For example:

   ```
   mqsisetdbparms INODE -n loopback::lbreqid1 -u myLoopBackUserID -p myLoopBackPassword
   ```

2. The specified credentials are passed to the LoopBack connector and are used to connect to a data source.

What to do next

Create a message flow and add a LoopBackRequest node to enable you to create, retrieve, update, or delete records through your LoopBack connector. For more information, see "Accessing and creating records by using the LoopBackRequest node" on page 1217.

Accessing and creating records by using the LoopBackRequest node

You can use the LoopBackRequest node to interact with the LoopBack framework, enabling you to create, retrieve, update, and delete records in backend data sources such as MongoDB, Cloudant, and PostgreSQL.

Before you begin

Complete the tasks in the following topics:

- "Installing the LoopBack connector" on page 1213
- "Configuring the data source and models for your LoopBack connector" on page 1214
- "Specifying security credentials for connecting to a secured data source" on page 1216

About this task

The following topics explain how to work with records in a client system by using a LoopBackRequest node:

- "Creating records by using a LoopBackRequest node" on page 1218
- "Retrieving records by using a LoopBackRequest node" on page 1219
- "Updating records by using a LoopBackRequest node" on page 1223
- "Deleting records by using a LoopBackRequest node" on page 1224

When you pass data to a data source through a LoopBackRequest node (when you create or update a record, for example), you must use the JSON domain for providing data in the input message. Replies to a LoopBack request are provided as JSON domain messages. For more information, see "Using models with LoopBack connectors" on page 1229.

Creating records by using a LoopBackRequest node
Use the LoopBackRequest node to create a new record in a backend data source by using a LoopBack connector.

About this task

You can use a LoopBackRequest node in a message flow to create a new data record through a LoopBack connector in a backend data source such as MongoDB, Cloudant, or PostgreSQL.

You create a new record by specifying `Create` in the `Operation` property of the LoopBackRequest node, and specifying the resource type (such as Account) in the `LoopBack object` property. The object can be overridden by the `LocalEnvironment.Destination.Loopback.Request.object` local environment variable. For more information about using environment variables, see "Using local environment variables with LoopBackRequest nodes" on page 1226.

The record that you are creating must exist in the input message tree as a JSON object. When the record is created, the input message tree is copied to the location specified by the `Output data location` property. The output data might also be augmented with additional data that was created in the backend data source.

Procedure

1. Create a flow containing a LoopBackRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Data source` property, specify the name of the data source stanza for your chosen connector in the LoopBack `datasources.json` file.
      - In the `Operation` property, specify `Create`.
      - In the `LoopBack object` property, specify the type of object (for example, `Account`) for which you are creating a new record.
      - In the `Security identity` property, specify the security identity that you will use when connecting to the LoopBack connector. For information about configuring the security identity on the integration node, see "Specifying security credentials for connecting to a secured data source" on page 1216.
      - In the `Timeout (milliseconds)` property, specify the time (in milliseconds) that the node waits for the LoopBack connector to process the operation.

      These properties can be overridden dynamically in the flow, as described in "Using local environment variables with LoopBackRequest nodes" on page 1226.
   b) On the **Request** tab, set the `Data location` property to specify the location in the incoming message tree that contains the JSON object data to be created in the external application; this data forms the request that is sent from the LoopBackRequest node to the connected system.
   c) On the **Result** tab, set the `Output data location` property to specify the location in the output message tree that will contain the data of the record that is created.

      Input and output messages are in the JSON domain. For more information, see "Combining a result message with an input message when fetching data from external systems" on page 1811.

   For more information about the properties of the LoopBackRequest node, see node.
2. Send the required message through the flow to the backend data source.

   You will receive a confirmation message, which might be augmented with additional data from the backend data source. The message will be in the JSON domain and can be created and processed by any suitable transformation node. To use the Mapping node, you must provide a message model of the data, as described in Creating or transforming a JSON message by using a JSON schema.
3. Check the backend data source to verify that the new record has been created correctly.

   You can also check the activity log for additional information about events related to the LoopBackRequest node. If the Loopback operation was not successful, the node reports by raising an exception with a BIP message that might include an error returned from the installed LoopBack

connector. If the failure terminal is connected, the node will propagate the exception list to it, otherwise the flow will roll back.

Retrieving records by using a LoopBackRequest node
Use a LoopBackRequest node in a message flow to retrieve data records through a LoopBack connector such as MongoDB, Cloudant, or PostgreSQL.

Before you begin

Complete the tasks in the following topics:

- "Installing the LoopBack connector" on page 1213
- "Configuring the data source and models for your LoopBack connector" on page 1214
- "Specifying security credentials for connecting to a secured data source" on page 1216

About this task

You retrieve records by specifying `Retrieve` in the `Operation` property of the LoopBackRequest node. You can retrieve all records of a particular type by specifying the object type in the `LoopBack object` property of the node. This object type can be overridden by a value specified in the `LocalEnvironment.Destination.Loopback.Request.object` local environment variable.

Alternatively, you can retrieve a specific record by specifying an ID in the `LocalEnvironment.Destination.Loopback.Request.id` environment variable. The returned record appears as a JSON object in the location that is specified by the `Output data location` property of the node.

If an external ID name and value are specified in the `LocalEnvironment.Destination.Loopback.Request.externalIdName` and `LocalEnvironment.Destination.Loopback.Request.externalId` environment variables, all records with that external ID are returned. The returned records appear as a JSON array in the location specified by the `Output data location` property.

You can control the records that are returned by using the `LocalEnvironment.Destination.Loopback.Request.filter` environment variable to specify a filter with the following clauses:

- `where`
- `limit`
- `skip`
- `order`
- `field`

If your installed LoopBack connector supports additional capabilities beyond those provided by these filter options, you can use the `filterString` local environment variable instead of `filter`. If both `filter` and `filterString` are specified, `filterString` takes precedence. For more information, see "Filtering the records retrieved through a LoopBack connector" on page 1220.

By default, the retrieval of records is determined by the following properties, in order of precedence:

- ID
- External ID
- Filter

If no records are found, the output tree contains nothing under JSON.data.

If you have configured a model for use with your LoopBack connector, see "Using models with LoopBack connectors" on page 1229, for additional information.

Procedure

Follow these steps to retrieve records by using a LoopBackRequest node:

1. Create a flow containing a LoopBackRequest node, and set the following properties on the **Basic** tab:
   a) In the `Data source` property, specify the name of the data source stanza for your chosen connector in the LoopBack `datasources.json` file.
   b) In the `Operation` property, specify `Retrieve`.
   c) In the `LoopBack object` property, select the type of object that you want to retrieve.
   d) In the `Security identity` property, specify the security identity that you will use when connecting to the LoopBack application. For information about configuring the security identity on the integration node, see "Specifying security credentials for connecting to a secured data source" on page 1216.
   e) In the `Timeout (milliseconds)` property, specify the time (in milliseconds) that the node waits for the LoopBack connector to process the operation.

   These properties can be overridden dynamically in the flow, as described in "Using local environment variables with LoopBackRequest nodes" on page 1226. For more information about the LoopBackRequest node properties, see node.
2. Send the required message through the flow to the LoopBack application.

   The node emits the JSON data of the record, which can be an empty message if no record is found. The message will be in the JSON domain and can be created and processed by any suitable transformation node. To use the Mapping node, you must provide a message model of the data, as described in Creating or transforming a JSON message by using a JSON schema.
3. Check that the records have been retrieved correctly.

   You can also check the activity log for additional information about events related to the LoopBackRequest node. If the Loopback operation was not successful, the node reports by raising an exception with a BIP message that might include an error returned from the installed LoopBack connector. If the failure terminal is connected, the node will propagate the exception list to it, otherwise the flow will roll back.

Filtering the records retrieved through a LoopBack connector
Use the **`filter`** element in the LocalEnvironment.Destination.Loopback.Request.filter environment variable to restrict the content of records that are retrieved from the LoopBack application.

About this task

You can restrict the content of records that are returned by using the `LocalEnvironment.Destination.Loopback.Request.filter` environment variable to specify a filter with a `where`, `limit`, or `skip` clause. You can also specify a filter with an `order` clause, and the records are returned in the specified order. The returned records appear as a JSON array or a JSON object in the location that is specified by the `Output data location` property of the LoopBackRequest node. By default, records are returned with all fields populated; however, you can use a filter with a `field` clause to restrict the data that is returned by explicitly selecting or excluding fields.

Examples are shown of how to set the values by using ESQL; however, you can also set them by using a transformation node such as a Mapping node. To access the LocalEnvironment from a Mapping node, see Customizing a message map to include a message assembly component.

By using the *LocalEnvironment.Destination.Loopback.Request.filter* environment variable, you can specify filter clauses, by using child elements of the **`filter`** element:

```
filter
     where <value>
     limit <value>
     skip <value>
     order (1..n)
          <fieldname>/<value>
     field (1..n)
          <fieldname>/<value>
```

You can specify the child elements of the **filter** element to restrict the content of the records that are retrieved. The available elements are shown in the following table:

Table 40. Elements used for filtering the content of records retrieved from a backend data source

Element	Type	Description
where	string	A condition expression used to filter the set of returned records.
limit	positive integer	Imposes a maximum number of records in the returned set.
skip	positive integer	Removes the first *n* records from the returned set before returning the number of results up to the specified `limit`.
order	structure	One child, the name of which must match the name of a field of the record. The value is either *ASC* or *DESC* (case-sensitive) and is used to order the returned set. You can specify multiple `order` clauses, which are applied in the order in which they appear in the `filter` tree.
field	structure	One child, the name of which must match the name of a field of the record. The value can be either *true* or *false* (or *1* or *0*), and controls whether the field is included as part of each returned record. You can specify multiple `field` clauses.

Examples

The following examples show some of the query filter options that you can use with the LoopBackRequest node.

The filtering options that are available to you depend on the capabilities of the specific LoopBack connector that you are using. The precise syntax and outcome of the filter also depend on the type and version of the installed LoopBack connector.

Get documents from MongoDB, where **price** is *>= 325* and **phone** begins with *08*:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"and": [{"price":
 {"gte":325}},
 {"phone": {"regexp": "^08"}}]}';
```

Get all records in descending **totalEmployees** order and ascending **companyName** order, retrieving only the **companyName** field, and explicitly excluding the **totalEmployees** field:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.order[1].totalEmployees = 'DESC';
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.order[2].companyName = 'ASC';
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.field[1].companyName = true;
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.field[2].totalEmployees = false;
```

- When you use a `field` clause that explicitly selects the set of fields to be returned, some connectors continue to return the **id** field unless it is explicitly excluded.

- You can use a model along with a **filter** and a `field` clause that explicitly selects the set of fields to be returned. When you use this option, some connectors return a `null` value for all field values that are not explicitly selected or excluded.

When you use a Mapping node to achieve this task, you can create the required fields by adding a user-defined element. For more information, see Adding a user-defined element.

When you use an ESQL Compute node to achieve this task, you must set the **Compute mode** to include *LocalEnvironment*.

Query by **id**, where the model defines the **identity field** as **companyId**. In this case, the **companyId** value is space-padded to match the length of a fixed-length character column in the backend database:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.id = '500  ';
```

Query by **externalId** and **externalIdName** by using **companyId** as the value for **externalIdName**. In this case, the **companyId** value is space-padded to match the length of a fixed-length character column in the backend database:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.externalId = '100  ';
SET OutputLocalEnvironment.Destination.Loopback.Request.externalIdName = 'companyId';
```

Query using **companyId** in a `filter.where` clause. In this case, the **companyId** value is space-padded to match the length of a fixed-length character column in the backend database:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"companyId":"400  "}';
```

Query using **totalEmployees** *>= 325* in a `filter.where` clause. Get two records and skip over the first one:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"totalEmployees":
 {"gte":325}}';
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.limit = 2;
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.skip = 1;
```

Query that uses **totalEmployees** *<= 430* and **companyName** begins with *B*:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filter.where = '{"and":
 [{"totalEmployees": {"lte":430}},
 {"companyName": {"like": "B%"}}]}';
```

Query that uses **filterString**, where **totalEmployees** is *<= 430* and **companyName** does not begin with *B*. Get only one record (`limit=1`) and ignore the first one (`skip=1`):

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filterString = '{"where": {"and":
 [{"totalEmployees":
 {"lte":430}}, {"companyName": {"nlike": "B%"}}]}, "limit":1, "skip":1}';
```

Query that uses **filterString** where **totalEmployees** is *<= 430* and **companyName** does not begin with *B*. Get the records in ascending **companyName** order and then descending **totalEmployees** order. Request the **companyName** field and explicitly exclude the **totalEmployees** field:

```
SET OutputLocalEnvironment.Destination.Loopback.Request.filterString = '{"where": {"and":
 [{"totalEmployees": {"lte":430}},
 {"companyName": {"nlike": "B%"}}]}, "order": ["companyName ASC", "totalEmployees DESC"],
 "fields": {"companyName": true,
 "totalEmployees": false}}';
```

Updating records by using a LoopBackRequest node
Use a LoopBackRequest node in a message flow to update data records through a LoopBack connector such as MongoDB, Cloudant, or PostgreSQL.

Before you begin

Complete the tasks in the following topics:

- "Installing the LoopBack connector" on page 1213
- "Configuring the data source and models for your LoopBack connector" on page 1214
- "Specifying security credentials for connecting to a secured data source" on page 1216

About this task

You can update records by specifying `Update` in the `Operation` property of the LoopBackRequest node, and specifying the object type in the `LoopBack object` property. The object type can be overridden by the `LocalEnvironment.Destination.Loopback.Request.object` environment variable.

The fields of the data record to be updated must appear in the input message tree as a JSON object. Fields that do not require updating do not have to appear in the tree.

If an ID is specified in the LocalEnvironment.Destination.Loopback.Request.Id environment variable, the record with that ID value is updated. If no record is found with a matching ID, an error is returned.

If an ID is not specified, but an external ID name and value are specified in the LocalEnvironment.Destination.Loopback.Request.externalIdName and LocalEnvironment.Destination.Loopback.Request.externalId environment variables, the record with that external ID is updated. In some cases, multiple records might match the same external ID. If no ID or external ID is specified, you can use a `filter.where` clause to specify the records to be updated. For more information about filtering, see "Filtering the records retrieved through a LoopBack connector" on page 1220.

When records are updated, data is returned from the installed LoopBack connector, such as a count of the updated records.

You can use JSON schema as input and output for message maps in a Mapping node, see "Using models with LoopBack connectors" on page 1229.

Procedure

Follow these steps to update a data record of a specified type (for example, an `Account` object):

1. Create a flow containing a LoopBackRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Data source` property, specify the name of the data source stanza for your chosen connector in the LoopBack `datasources.json` file.
      - In the `Operation` property, specify `Update`.
      - In the `LoopBack object` property, select the type of object that you want to update (for example, `Account`).
      - In the `Security identity` property, specify the security identity that you will use when connecting to the LoopBack connector. For information about configuring the security identity

on the integration node, see "Specifying security credentials for connecting to a secured data source" on page 1216.

- In the `Timeout (milliseconds)` property, specify the time (in milliseconds) that the node waits for LoopBack connector to process the operation.

These properties can be overridden dynamically in the flow, as described in "Using local environment variables with LoopBackRequest nodes" on page 1226.

b) On the **Request** tab, set the `Data location` property to specify the location in the incoming message tree that contains the JSON objects data that will be used to update the data record.

For more information, see node.

2. Ensure that the ID of the record that you want to update is set in the LocalEnvironment.Destination.Loopback.Request.id local environment variable.

   Alternatively, you can update records by specifying the appropriate values in the LocalEnvironment.Destination.Loopback.Request.externalId and LocalEnvironment.Destination.Loopback.Request.externalIdName local environment variables.

3. Send the required message through the message flow to the LoopBack application.

   You will receive a response from the application to confirm that the records have been updated.

4. Check the backend data source to verify that the records have been updated correctly.

   You can also check the activity log for additional information about events related to the LoopBackRequest node. If the Loopback operation was not successful, the node reports by raising an exception with a BIP message that might include an error returned from the installed LoopBack connector. If the failure terminal is connected, the node will propagate the exception list to it, otherwise the flow will roll back.

Deleting records by using a LoopBackRequest node

Use a LoopBackRequest node in a message flow to delete a data record through a LoopBack connector such as MongoDB, Cloudant, or PostgreSQL.

Before you begin

Complete the tasks in the following topics:

- "Installing the LoopBack connector" on page 1213
- "Configuring the data source and models for your LoopBack connector" on page 1214
- "Specifying security credentials for connecting to a secured data source" on page 1216

About this task

You delete a record by specifying `Delete` in the `Operation` property of the LoopBackRequest node, and specifying the ID of the record in the LocalEnvironment.Destination.Loopback.Request.id environment variable. If no record is found with a matching ID, an error is returned.

If no ID is specified, but an external ID name and value are specified in the LocalEnvironment.Destination.Loopback.Request.externalIdName and LocalEnvironment.Destination.Loopback.Request.externalId environment variables, the records with that external ID are deleted. In some cases, multiple records might match the same external ID. If no ID or external ID is specified, you can use a `filter.where` clause to specify the records to be deleted. For more information about filtering, see "Filtering the records retrieved through a LoopBack connector" on page 1220.

When records are deleted, a result is returned from the installed connector, such as a count of the deleted records.

Procedure

Follow these steps to delete a record:

1. Create a flow containing a LoopBackRequest node, and set the node properties:
   a) On the **Basic** tab, set the following properties:
      - In the `Data source` property, specify the name of the data source stanza for your chosen connector in the LoopBack `datasources.json` file.
      - In the `Operation` property, specify `Delete`.
      - In the `LoopBack object` property, select the type of object that you want to delete.
      - In the `Security identity` property, specify the security identity that you will use when connecting to the LoopBack application. For information about configuring the security identity on the integration node, see "Specifying security credentials for connecting to a secured data source" on page 1216.
      - In the `Timeout (milliseconds)` property, specify the time (in milliseconds) that the node waits for the waits for the LoopBack connector to process the operation.

      These properties can be overridden dynamically in the flow, as described in "Using local environment variables with LoopBackRequest nodes" on page 1226.
   b) On the **Results** tab, set the `Output data location` property to control where the JSON object that contains the count of the deleted records will be placed in the output message tree.

   For more information, see node.
2. Ensure that the ID of the record that you want to delete is set in the LocalEnvironment.Destination.Loopback.Request.id local environment variable.

   You can also delete records by specifying values in the LocalEnvironment.Destination.Loopback.Request.externalId and LocalEnvironment.Destination.Loopback.Request.externalIdName local environment variables.
3. Send the required message through the flow to the LoopBack application.

   If the delete operation is successful, the output tree will contain the count of the deleted records.
4. Go to your account in the backend data source to verify that the record has been deleted successfully.

   You can also check the activity log for additional information about events related to the LoopBackRequest node. If the Loopback operation was not successful, the node reports by raising an exception with a BIP message that might include an error returned from the installed LoopBack connector. If the failure terminal is connected, the node will propagate the exception list to it, otherwise the flow will roll back.

Scaling throughput of IBM App Connect Enterprise message flow applications that use LoopBackRequest nodes

Due to the single threaded event loop mechanism that exists in the Node.js architecture, you might need to adopt a specific approach to performance scaling when you are running IBM App Connect Enterprise message flow applications that useLoopBackRequest nodes. You might find that you do not achieve the scaling that you require just by increasing the additional instances on these message flows. If that is the case, you can continue to scale performance by running instances of such applications in separate integration servers.

The IBM App Connect Enterprise LoopBackRequest node exploits the industry standard Node.js framework and enables message flows to connect with external systems by using user-installed LoopBack connectors.

Node.js architecture uses the single threaded event loop model that manages the requests and responses of all worker threads by using a non-blocking callback mechanism. In IBM App Connect Enterprise, each integration server is a separate process and, therefore, can host a single Node.js instance. As a consequence, all message flow threads in an integration server must send LoopBackRequest node requests and responses to the external system by using the single threaded event loop in Node.js. These threads then wait for the callback to be sent back through the single threaded event loop after the required action has been completed by the interaction of the LoopBack connector worker thread and the external system.

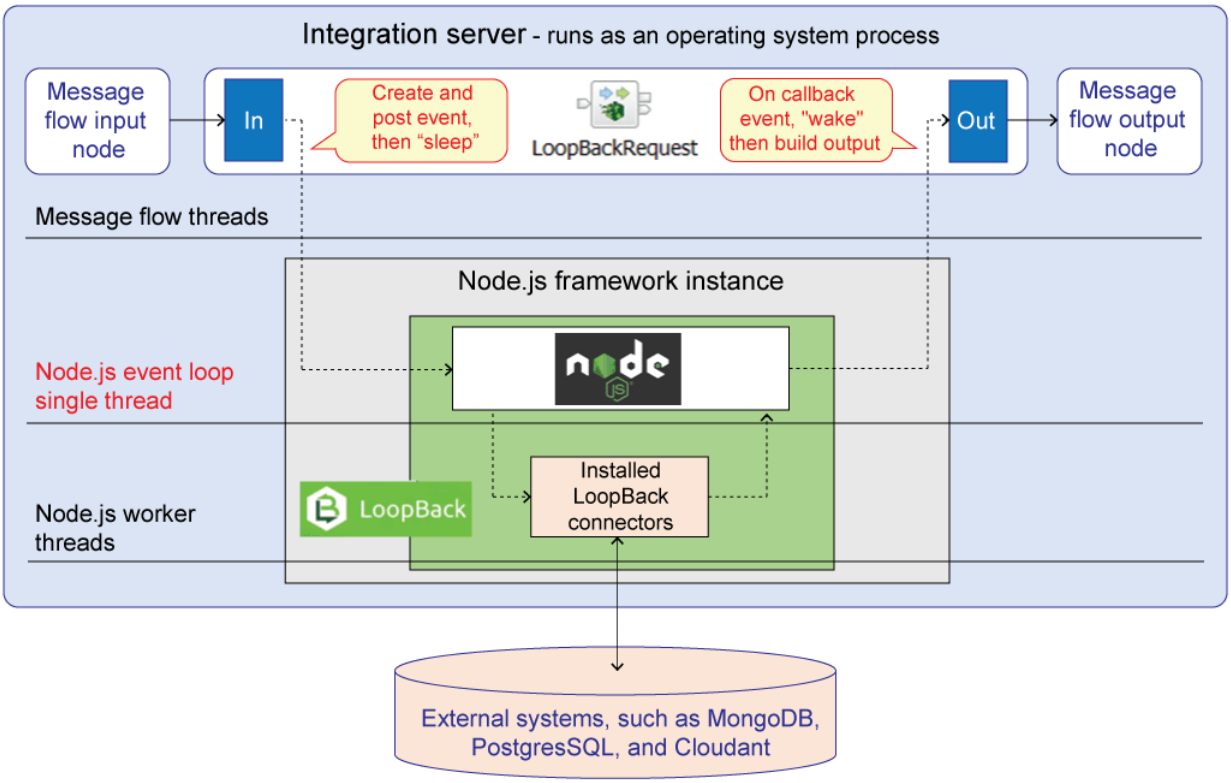


Figure 3. The flow of activity between an IBM App Connect Enterprise integration server that is using LoopBackRequest nodes to interact with an external system.

To achieve throughput scaling of IBM App Connect Enterprise message flow applications that use LoopBackRequest nodes, start by increasing the number of thread instances that are available to the message flow application in a single integration server. Use the **Additional instances** property on the message flow input node or set the workload management **Additional instances** property of the message flow to set the number of message flow threads that are available. If further increases to the **Additional instances** property no longer scale the throughput, run instances of the message flow application in separate integration servers.

Using local environment variables with LoopBackRequest nodes

The LoopBackRequest node supports a number of local environment message tree variables, which you can use to dynamically alter the values that are set in the node properties.

The following table shows the environment variables that can be set to control the behavior of the LoopBackRequest node. The table includes examples of how to set the value by using ESQL; however, you can also set them by using transformation nodes, such as a Mapping node. To access the LocalEnvironment from a Mapping node, see Customizing a message map to include a message assembly component.

Local environment variable	Type	Description
LocalEnvironment.Destination.Loopback.Request.operation	string	The operation to be requested on the external system: • `Create` • `Update` • `Retrieve` • `Delete` This environment variable overrides the **`Operation`** property on the LoopBackRequest node.
LocalEnvironment.Destination.Loopback.Request.object	any	The object for the request. This environment variable overrides the **`LoopBack object`** property on the LoopBackRequest node, and applies for all operations.
LocalEnvironment.Destination.Loopback.Request.id	string	The LoopBack object ID of the record to be accessed by this request. This value is used for the following operations: • `Retrieve` (optional) This environment variable causes a single record to be retrieved, and takes precedence over the `externalId`, `filter`, and `filterString` environment variables. • `Update` (optional) This environment variable causes a single record to be updated, and takes precedence over the `externalId` environment variable. If this environment variable is not set, either the `externalId` or `filter.where` environment variable must be set. • `Delete` (optional) This environment variable causes a single record to be deleted, and takes precedence over the `externalId` environment variable. If this environment variable is not set, either the `externalId` or `filter.where` environment variable must be set. This `Id` is assigned to a record when it is created on the external system.

Local environment variable	Type	Description
LocalEnvironment.Destination.Loopback.Request.externalIdName	string	The name of an external ID field of a record in the external system. This value is used for the following operations: • `Retrieve` (optional) Causes a set of records that have the field name specified in this environment variable to be retrieved. • `Update` (optional) Causes a set of records that have the field name specified in this environment variable to be updated. • `Delete` (optional) Causes a set of records that have the field name specified in this environment variable to be deleted.
LocalEnvironment.Destination.Loopback.Request.externalId	string	The external ID of a record in the external system. This value is used for the following operations: • `Retrieve` (optional) Causes a set of records that have the external ID values given in this environment variable to be retrieved. This variable takes precedence over `filter.where` and `filterString` environment variables. • `Update` (optional) Causes a set of records that have the external ID values given in this environment variable to be updated. If neither the `Id` nor `filter.where` environment variable is set, this variable is required. • `Delete` (optional) Causes a set of records that have the external ID values given in this environment variable to be deleted. If neither the `Id` nor `filter.where` environment variable is set, this variable is required.

Local environment variable	Type	Description
LocalEnvironment.Destination.Loopback.Request.filterString	String	A pre-built string format of the JSON filter object for this request. Used only for `Retrieve` (optional). You can use it to set any filter that the connector supports, for example: `{"where": {"and": [{"thing1":"isOne"}, {"thing2":"isTwo"}]}}` If `filterString` and `filter.where` are both specified, only `filterString` is used.
LocalEnvironment.Destination.Loopback.Request.filter	Structure	The filter to be used for restricting the records acted upon by an operation. • `Retrieve` (optional) Restricts the records returned by a `Retrieve` operation. You can specify the following sub-fields to set parts of the JSON filter object for this request: – `where` – `limit` – `skip` – `order` – `field` • `Update` (optional) Restricts the records updated by an `Update` operation to those that are specified by the `where` sub-field. • `Delete` (optional) Restricts the records deleted by a `Delete` operation to those that are specified by the `where` sub-field. For more information, see "Filtering the records retrieved through a LoopBack connector" on page 1220.
LocalEnvironment.Destination.Loopback.Request.timeoutMilliseconds	Integer	The time (in milliseconds) that the node waits for the external system to process the operation. This environment variable overrides the **`Timeout (milliseconds)`** property on the LoopBackRequest node.

Using models with LoopBack connectors

Interactions between the LoopBackRequest node and some types of backend data source require LoopBack models to be defined.

Some LoopBack connectors require that the data model to be interacted with is defined through a LoopBack JSON model. When the LoopBackRequest node interacts for the first time with a named LoopBack object (which is either specified as a node property or dynamically overridden in the local

environment), it looks for a LoopBack JSON model file in a subdirectory of the directory that contains the `datasources.json` file. For information about configuring the model, see "Configuring the data source and models for your LoopBack connector" on page 1214.

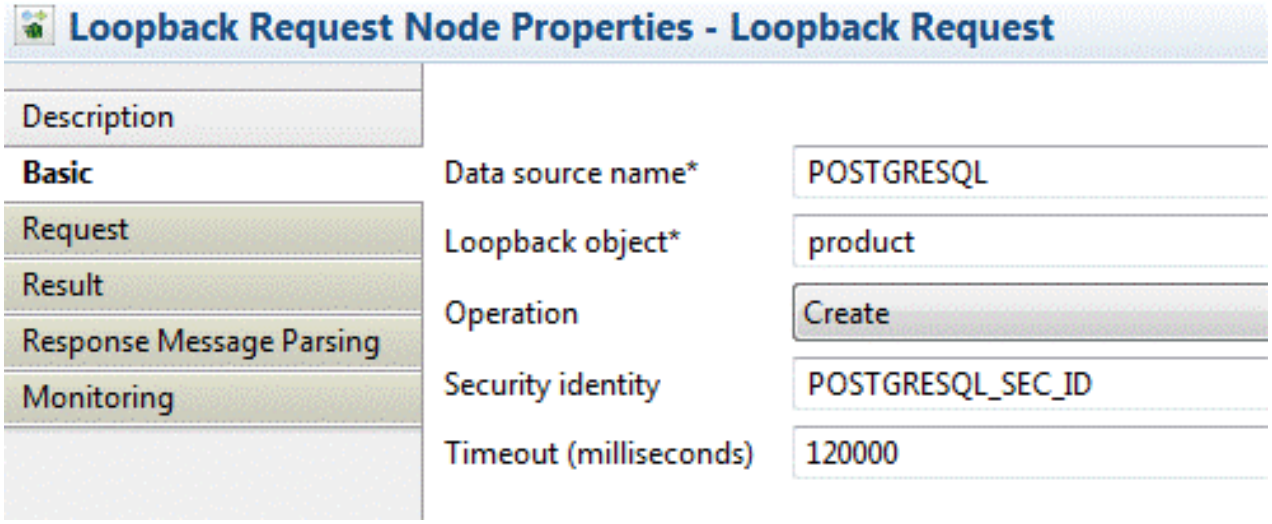
A LoopBack model is typically required for data sources that are backed by relational databases, where table objects with a rigid data structure are used. With these types of data source, the model can be used to define property mappings between the database columns and the LoopBack properties that are used by the LoopBackRequest node.

In the following example, the integration node called `IIB10` uses a `datasources.json` file that is located in the *MQSI_WORKPATH*`\IIB10\connectors\loopback` directory, where *MQSI_WORKPATH* is defined by the MQSI_WORKPATH environment variable. The `datasources.json` file includes the following stanza, called POSTGRESQL, to connect to the PostgreSQL database:

```
"POSTGRESQL": {
    "host": "localhost",
    "port": 5432,
    "database": "loopback",
    "name": "postgreSQL",
    "connector": "postgresql"
  }
```

The `product.json` file is copied into the *MQSI_WORKPATH*`\IIB10\connectors\loopback\POSTGRESQL` directory. For the LoopBackRequest node to use the model, the **LoopBack object** property on the node (or the LocalEnvironment.Destination.Loopback.Request.object variable, if used) must match exactly the name of the model, excluding the `.json` file extension. Typically, these names are the same, but they can be different; for example, if the name of the object in the LoopBack connected system contains a character that would be invalid in a file name. If the names are different, a BIP3880 message is issued.

In this case, the **LoopBack object** property is configured as shown in the following diagram:



The following example shows how to use a LoopBack model to interact with a `product` table that is defined in a PostgreSQL database:

```
CREATE TABLE public.product
(
  product_id serial NOT NULL,
  product_name character varying(20),
  sell_by date,
  number_in_stock integer,
  obsolete boolean,
  CONSTRAINT product_pk PRIMARY KEY (product_id)
)
```

The LoopBack PostgreSQL connector supports discovery, so the StrongLoop® and API Connect model-generation tools were used to generate the following LoopBack model for the `product` table. This model was output to the `product.json` file:

```
{
  "name": "Product",
  "base": "PersistedModel",
  "idInjection": false,
  "options": {
    "validateUpsert": true
  },
  "postgresql": {
    "schema": "public",
    "table": "product"
  },
  "properties": {
    "numberInStock": {
      "type": "number",
      "required": false,
      "length": null,
      "precision": 32,
      "scale": 0,
      "postgresql": {
        "columnName": "number_in_stock",
        "dataType": "integer",
        "dataLength": null,
        "dataPrecision": 32,
        "dataScale": 0,
        "nullable": "YES"
      },
      "_selectable": true,
      "comments": "Items in stock"
    },
    "obsolete": {
      "type": "boolean",
      "required": false,
      "length": null,
      "precision": null,
      "scale": null,
      "postgresql": {
        "columnName": "obsolete",
        "dataType": "boolean",
        "dataLength": null,
        "dataPrecision": null,
        "dataScale": null,
        "nullable": "YES"
      },
      "_selectable": true,
      "comments": "Flag if stock is obsolete"
    },
    "productId": {
      "type": "number",
      "id": true,
      "required": true,
      "length": null,
      "precision": 32,
      "scale": 0,
      "postgresql": {
        "columnName": "product_id",
        "dataType": "integer",
        "dataLength": null,
        "dataPrecision": 32,
        "dataScale": 0,
        "nullable": "NO"
      },
      "_selectable": false,
      "comments": "Unique product number"
    },
    "productName": {
      "type": "string",
      "required": true,
      "length": 20,
      "precision": null,
      "scale": null,
      "postgresql": {
        "columnName": "product_name",
        "dataType": "character varying",
        "dataLength": 20,
        "dataPrecision": null,
        "dataScale": null,
```

```
        "nullable": "YES"
      },
      "_selectable": true,
      "comments": "Description of product"
    },
    "sellBy": {
      "type": "date",
      "required": false,
      "length": null,
      "precision": null,
      "scale": null,
      "postgresql": {
        "columnName": "sell_by",
        "dataType": "date",
        "dataLength": null,
        "dataPrecision": null,
        "dataScale": null,
        "nullable": "YES"
      },
      "_selectable": true,
      "comments": "Expiry date"
    }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": {}
}
```

For more information about the StrongLoop and API Connect model-generation tools, see the documentation on the StrongLoop website.

In the PostgreSQL `product` table, the `product_id` column is defined with a data type of `serial`, which results in values for the column being generated automatically when a row is inserted into the table. To reflect this, the LoopBack `product.json` model has been updated to include a `generated` property for the `productId`. The `required` attribute has also been changed to `false`, which stops the model checking that the data in requests sent from the LoopBackRequest node includes a `productId` when a row is created, because this value will be generated automatically. The updated model is shown in the following example:

```
"productId": {
      "type": "number",
      "id": true,
      "required": false,
      "generated": true,
      "length": null,
      "precision": 32,
      "scale": 0,
      "postgresql": {
        "columnName": "product_id",
        "dataType": "integer",
        "dataLength": null,
        "dataPrecision": 32,
        "dataScale": 0,
        "nullable": "NO"
      },
      "_selectable": false,
      "comments": "Unique product number"
    }
```

The model can also be used to set default values for data values; in the following example, the `obsolete` value defaults to `false`:

```
"obsolete": {
      "type": "boolean",
      "required": false,
      "default": false,
      "length": null,
      "precision": null,
```

Routing messages

Route messages through your message flow by using one or more of the techniques described in this section.

About this task

- “Using nodes for decision making” on page 1233
- “Routing using publish/subscribe applications” on page 1235
- “Using Operational Decision Manager (ODM) business rules” on page 1241

Some nodes use fields in your messages to make routing decisions. You might need to define a model of your messages to enable access to the fields in the message for these nodes. For information about how to develop a message model, and why you might want to develop a message model, see “Constructing message models” on page 2133.

Using nodes for decision making

You can use several built-in nodes in different ways to control the path that a message takes through the message flow.

Before you begin

Read the concept topic about message flow nodes.

About this task

These nodes let you decide how messages are processed by specifying the route that each message takes through the message flow based on dynamic values such as message structure and content.

For more information, see the following topics:

- “Testing the message structure (Validate node)” on page 1233
- “Controlling the order of processing in a message flow” on page 1234
- “Using the destination list to route messages (RouteToLabel and Label nodes)” on page 1235

You can use one of the following nodes to determine the path taken by a message through the message flow based on its content:

- node
- Filter node (ESQL)
- JavaCompute node
-
- Windows node
- node

You can use any of the following nodes to route messages by using information from a database:

- node
- node

Testing the message structure (Validate node)

Use the Validate node to test the characteristics of the message structure.

About this task

If you set the Validate node properties appropriately, you can request that one or all of the message domain, message set, and message type are compared to a specific value. If the message matches those

values for which you have requested the check, it is routed through the match terminal and is processed by the sequence of nodes that you have connected to that terminal.

If the message does not match any one of those values for which you have requested the check, it is routed through the failure terminal and is processed by the sequence of nodes that you have connected to that terminal.

For example, you might design a message flow that provides additional processing for all messages that are in the MRM domain. You can include a Validate node that tests just that characteristic of the message, and passes it to a sequence of nodes that provide the specialized processing. If the message is not in the MRM domain, the extra nodes are bypassed, and the failure terminal is wired up directly to the node that follows the sequence required for MRM messages only.

Controlling the order of processing in a message flow

Use the FlowOrder node to control the order of processing in a message flow.

About this task

When you connect message flow nodes together, the integration node determines the way in which the different connections are processed. This includes the order in which they are processed. If you have connected more than one message flow node or sequence of nodes to a single output terminal, you cannot predict whether one sequence is processed before another for a message.

If the order of processing is important in your message flow, use the FlowOrder node to force a prescribed order of processing of the messages that are propagated by this node.

The FlowOrder node has two output terminals that you can connect to control the order in which subsequent nodes process the message. The output terminals, named First and Second, are always processed in that order.

When you connect a message flow node or sequence of nodes to the First terminal, the input message is passed to the next node, and all processing defined by all subsequent nodes in this sequence is completed before control returns to the FlowOrder node.

The input message is then propagated to the next node in the sequence of nodes connected to the Second terminal.

The message passed to both sequences of nodes, from the First terminal and the Second terminal, is identical. It is always the message that the FlowOrder node receives as input. The message that the FlowOrder node propagates to the Second terminal is in no way affected by the processing of the message that has been performed by the sequence of nodes connected to the First terminal.

The FlowOrder node provides no other processing on the input message; it is used only for imposing order on subsequent processing.

Testing the message content (Filter node)

You can use the Filter node to determine the path taken by a message through the message flow based on its content.

About this task

You can customize the Filter node by using ESQL statements to determine if the message content meets some condition. The condition tested must yield a Boolean result, that is it must be true or false (or unknown). You can create the test to reference information from a database, if applicable.

You can connect nodes following the Filter node to the corresponding terminals of the Filter node, and process the message according to its content.

Using the destination list to route messages (RouteToLabel and Label nodes)

You can determine the path that a message takes through the message flow by using the RouteToLabel and Label nodes.

About this task

These nodes provide a more flexible way to process messages than the Filter node, which depends on the Boolean result of an ESQL expression for its logic.

When you use RouteToLabel and Label nodes, you must include a Compute node that determines, by using some combination of message content, database content, and ESQL logic, how messages are to be processed next. Configure the Compute node to create a destination list (in the DestinationList folder in the local environment subtree) that contains the destination for each message, specified as the LabelName of a Label node. The Compute node passes the message to the RouteToLabel node, which reads the destination list and propagates the message to either the first or last item on the destination list, according to the value that is specified for the RouteToLabel node's Mode property. Although there is no limit to the number of destinations that the Compute node writes in the destination list, the RouteToLabel node propagates the message only to a single label node. This use of the destination list is in contrast to its use to define the final recipients of the output messages. For more information about the procedure for creating a destination list, see “Creating destination lists” on page 610.

If you intend to derive destination values from the message itself, or from a database, you might also need to cast values from one type to another. For more information about the local environment, see “Local environment tree” on page 509. For more information about casting, see Supported casts.

Routing using publish/subscribe applications

You can route your messages to applications using the publish/subscribe method of messaging.

In IBM App Connect Enterprise, you can send publication messages to publish/subscribe brokers, and receive messages from them by using subscriptions. There are two situations where you can achieve additional functionality by using IBM App Connect Enterprise for publish/subscribe. These situations are where you want to:

- Provide additional transformation or routing function, or both, at publication time
- Filter messages based on the content of the body of the message

Developing publish/subscribe applications

For background information about the publish/subscribe method of messaging, see the following topics:

- “Publish/subscribe overview” on page 1236
- “Publishers” on page 1236
- “Publications” on page 1236
- “Subscribers” on page 1237
- “Filters” on page 1238
- “Subscription points” on page 1238

Publish/Subscribe in IBM App Connect Enterprise 12.0

In IBM App Connect Enterprise 12.0, publish/subscribe functions are provided by the MQTT and Publication (IBM MQ) message flow nodes. IBM App Connect Enterprise uses publish/subscribe to notify applications of significant events that occur in integration nodes. IBM App Connect Enterprise provides a built-in MQTT broker to support these integration node events. The MQTT broker is enabled by default for all events, apart from monitoring events. If you have installed IBM MQ, you can also use the IBM MQ pub/sub broker in place of, or alongside, the built-in MQTT broker. If you have a queue manager that is specified on the integration node, the MQ pub/sub broker is enabled by default for all types of

event, including monitoring events. You can choose which type of pub/sub broker to use based on your processing requirements and your existing architecture.

Publish/subscribe overview

Publish/subscribe is a style of messaging application in which the providers of information (publishers) have no direct link to specific consumers of that information (subscribers), but the interactions between publishers and subscribers are controlled by pub/sub brokers.

In a publish/subscribe system, a publisher does not need to know who uses the information (publication) that it provides, and a subscriber does not need to know who provides the information that it receives as the result of a subscription. Publications are sent from publishers to the pub/sub broker, subscriptions are sent from subscribers to the pub/sub broker, and the pub/sub broker forwards the publications to the subscribers.

This style of messaging contrasts with a point-to-point style of messaging application, in which the application that sends messages needs to know the destinations of the messages that it sends.

The pub/sub broker ensures that messages are delivered to the correct subscribers. IBM App Connect Enterprise supports two types of pub/sub broker (MQTT and MQ), and you can choose which type to use based on your processing requirements and your existing architecture. A built-in MQTT broker is provided with IBM App Connect Enterprise, and MQTT publication is enabled by default for all events that are generated by the integration node, apart from business events. You can specify an alternative MQTT broker if you choose not to use the built-in MQTT broker. If you have installed IBM MQ, you can use the MQ pub/sub broker that is provided with IBM MQ.

A typical publish/subscribe system has more than one publisher and more than one subscriber. An application can be both a publisher and a subscriber.

The publisher generates a message that it wants to publish and defines the topic of the message. A subscriber registers a request for a publication by specifying the topic (or topics) of the published messages that it is interested in. Subscriptions to an MQ pub/sub broker might include the following information:

- The subscription point from which it wants to receive publications.
- The content filter that should be applied to the published message.
- The name of the queue (known as the subscriber queue) on which publications that match the criteria selected are placed. This queue can be a cluster queue, so that publications can be distributed to clustered subscribers.

Publishers

A *publisher* is an application that makes information about a specified topic available to a pub/sub broker in a publish/subscribe system. The pub/sub broker can be an MQ broker (queue manager) or an MQTT broker.

In a publish/subscribe system, an application, known as the publisher, can send a message to a pub/sub broker. Another application, known as the subscriber, can send a subscription request to the pub/sub broker, which then sends relevant publication messages to the message queue or port of the subscriber.

A published message can be requested by more than one subscriber, and a subscriber can request messages, on the same or different topics, from more than one publisher.

Publications

A *publication* is a piece of information about a specified topic that is available to a pub/sub broker in a publish/subscribe system. The pub/sub broker can be an MQ broker (queue manager) or an MQTT broker.

Typically, a pub/sub broker distributes a publication that it receives to all applications that have registered a subscription for the publication. If the pub/sub broker is an MQ broker (queue manager), it also distributes the publication to all other queue managers connected to it, either directly or through a network of queue managers that have subscribers for the publication.

Local publications

If your application uses an MQ pub/sub broker, publishers can restrict access to their publications to only those subscribers that are registered to the same pub/sub broker as the publisher. This publication is known as a *local publication*.

If you are using an MQTT broker, all publications are local. Local publications are not forwarded to other pub/sub brokers.

Global publications

A publication whose distribution is not restricted to subscribers that are registered to the same pub/sub broker as the publisher is known as *global publication*. Global publications can be published through MQ pub/sub brokers (queue managers) only. A global publication is forwarded to all MQ pub/sub brokers, connected either directly or through a network of queue managers, that have one or more subscribers for the publication.

State and event information

Information being published can be categorized either as *state* information or as *event* information.

State information is information about the current state of something. The current price of stock or the current score in a soccer match are both examples of state information.

Event information is information about an individual event that occurs. A change in the price of stock or the scoring of a particular goal in a soccer match are both examples of event information.

When an event occurs, the current state information is no longer required and is superseded by new state information.

If a publication contains state information, it is often published as a retained publication. A new subscriber typically wants the current information immediately; the subscriber does not want to wait for an event that causes the information to be republished.

Retained publications

Typically, a pub/sub broker discards a publication after it has been delivered to subscribers. However, a publisher can specify (in the case of the *Publish* message, by specifying the *RetainPub* option) that it wants the pub/sub broker to keep a copy of the publication, which is then called a *retained publication*.

If a retained publication has been published, new subscribers to that publication receive the publication without having to wait for it to be published again.

For example, a subscriber that registers a subscription for a stock price receives the latest published stock price immediately, and does not have to wait for the stock price to be republished.

An MQTT pub/sub broker retains only one publication for each topic. An MQ pub/sub broker retains only one publication for each combination of topic and subscription point.

Subscribers

A *subscriber* is an application that requests information about a specified topic from a pub/sub broker. A pub/sub broker can be either an MQ pub/sub broker (queue manager) or an MQTT broker.

The subscribing application might be an IBM App Connect Enterprise, IBM MQ, or MQTT client application.

The subscriber sends a subscription request to a pub/sub broker (MQTT or MQ), specifying which publications it wants to receive. The request defines the topic for each publication, and, for IBM MQ, it also defines the filter and the subscription point.

Messages that are published by a publisher can be received by more than one subscriber, and a subscriber can receive messages, on the same or different topics, from more than one publisher.

Subscription topics can also include wildcards, allowing them to match a range of related topics.

Filters

A *filter* is an expression, which might include wildcard characters, that is applied to the content of a publication message to determine whether it matches a subscription.

Filters are supported only if your application is using an MQ pub/sub broker; they are not supported by MQTT brokers.

When you register a subscription, in addition to specifying a topic and subscription point, you can specify a filter to select publications according to their contents. In the IBM MQ subscription properties, filters are referred to as *selectors*. For more information, see the IBM MQ product documentation online. IBM App Connect Enterprise must know how to parse the contents of the message correctly, which can be achieved in a number of ways:

- The message is a self-defining XML message.
- The message template is defined in the MQRFH2 header.

  If the message has an MQRFH header, the message set and type are taken from that header. Otherwise, the message is assumed to be as defined in the properties (domain, set, type, and format) of the input node.

The filter itself is entered as an ESQL expression; for example:

```
Body.Name LIKE 'Smit%'
```

This example means that the contents of a field called Name in the body of a publication message are extracted and compared with the string given in the expression. If the string in the message starts with the characters "Smit", the expression evaluates to TRUE and the publication is sent to the subscriber.

If you want to select publications using filters only, without specifying a topic, you can register a subscription with the required filter and a topic of "#" (all topics). You then receive publications on only those topics for which you have access authority.

Subscription points

A *subscription point* is the name that a subscriber uses to request publications from a particular set of Publication nodes. It is the property of a Publication node that differentiates that Publication node from other Publication nodes in the same message flow.

A subscriber that registers a subscription without specifying a subscription point receives publications from any unnamed Publication node in the message flow, if there is a match with the topic and filter specified by the subscriber. Subscription points are valid only if your application is emitting to an MQ pub/sub broker; they are not supported by MQTT brokers.

This behavior applies to all message flows running in all integration nodes connected in the same network, unless the local has been specified when registering the subscription.

Using subscription points

If you have more than one Publication node in a message flow, you can differentiate between them by specifying subscription points. Choose values that indicate the type of message that is routed to each Publication node.

Example

Consider an application that publishes stock prices. The prices that are available from the first Publication node in the message flow are in dollars. This Publication node uses the default subscription point.

You can define a second path through the message flow that takes the price in dollars, and converts this using a defined conversion value, to produce the same message but with the stock price in pounds. These messages are published at a second Publication node that has its subscription point property set to *Pounds*.

You might have another message flow, in the same integration node or a connected integration node, that publishes stock prices in pounds on the same topic. Make sure that it uses the *Pounds* subscription point,

and that all other message flows that publish their stock prices in dollars use the default subscription point.

Subscribers specifying the relevant topic (for example, *stock*) can then choose whether to receive the information in dollars or in pounds, by using the default subscription point or the *Pounds* subscription point when they subscribe.

Content-based filtering using ESQL

IBM MQ supports message selection based on `Root.MQMD` and message properties; IBM App Connect Enterprise provides extra options.

Content based filters in IBM App Connect Enterprise extend IBM MQ message selectors to allow the use of ESQL expressions to filter on the entire message.

You can use the following items:

Correlation names

In content based filters, the message can be accessed using the `Root`, `Body`, and `Properties` correlation names.

Any other correlation name is interpreted as an IBM MQ message property, apart from the following list of excluded reserved names:

- `Database`
- `DestinationList`
- `Environment`
- `LocalEnvironment`
- `ExceptionList`
- `LocalEnvironment`

Any other top level identifier is assumed to be an IBM MQ message property, and is accepted.

Trees

The content based filtering engine does not update the message tree. It accesses the message tree as read-only. Any statements that attempt to modify the tree are rejected.

By supporting ESQL, content based filtering in IBM App Connect Enterprise supports fully-qualified Namespaces, as documented in ESQL reference.

ESQL expressions

Content based filtering accepts any ESQL expression, with the exclusion of database access, that returns a Boolean value.

Enabling content-based filtering with publish/subscribe

IBM MQ provides the ability to specify a filter when a subscription is made, but this can only refer to items in headers. IBM App Connect Enterprise can act as a content filtering provider for IBM MQ, and so allows extended filters to be specified by subscribers that can refer to elements in the body of publications.

Before you begin

- Read about Selectors on IBM MQ.
- Read the following overview of how IBM MQ selects content of messages.

  When an application publishes on a topic string, where one or more subscribers have a selection string selecting on the content of the message, IBM MQ requests that the extended message selection

provider parse the publication and inform IBM MQ whether the publication matches the selection criteria specified by each subscriber with a content filter.

If the extended message selection provider determines that the publication matches the subscriber's selection string, the message continues to be delivered to the subscriber.

If the extended message selection provider determines that the publication does not match, the message is not delivered to the subscriber. This might cause the IBM MQ MQPUT or MQPUT1 call to fail with reason code MQRC_PUBLICATION_FAILURE. If the extended message selection provider is unable to parse the publication, reason code MQRC_CONTENT_ERROR is returned and the MQPUT or MQPUT1 call fails.

If the extended message selection provider is unavailable or is unable to determine whether the subscriber should receive the publication, reason code MQRC_SELECTION_NOT_AVAILABLE is returned and theIBM MQ MQPUT or MQPUT1 call fails.

When a subscription is being created with a content filter and the extended message selection provider is not available, the IBM MQ MQSUB call fails with reason code MQRC_SELECTION_NOT_AVAILABLE. If a subscription with a content filter is being resumed and the extended message selection provider is not available, the IBM MQ MQSUB call returns a warning of MQRC_SELECTION_NOT_AVAILABLE, but the subscription is allowed to be resumed.

- Each queue manager that supports Extended Message Selection (Content Based Filtering), requires a broker that is associated with it and Content Based Filtering enabled on at least one integration server. If a client subscribes and specifies a content-based filtering expression against a queue manager that does not have this function enabled, it receives MQRC_SELECTION_NOT_AVAILABLE on the subscription request.
- Linux Windows UNIX On Linux, UNIX and Windows systems, grant authorization for the system to the queue manager: `setmqaut -t <qmgr> +system -p <brokerUserId>`

About this task

IBM App Connect Enterprise extends the message selection support provided by IBM MQ. IBM App Connect Enterprise does this by allowing ESQL statements rather than SQL92 statements, and filtering based on message content. See “Content-based filtering using ESQL” on page 1239 for details of the scope and exclusions of supported ESQL.

The main external differences in the current implementation of content-based filtering are:

- Content based filtering is no longer limited to IBM MQ MQRFH2 subscribers. IBM App Connect Enterprise provides content filtering services for the following IBM MQ subscribers:

  MQRFH2
  MQSUB

  If you are performing content filtering based on the *NameValueData* field, within an MQRFH2 header, on z/OS, the data can be present either in the first or second MQRFH2 header. For example, the filter:

  ```
  Root.MQRFH2.mcd.Msd='XML'
  ```

  might not work as you expect on z/OS. Use the following syntax to search all MQRFH2 headers:

  ```
  FOR ANY Root.MQRFH2[] AS I (I.mcd.Msd='XML')
  ```

- IBM MQ message properties are supported as part of the filter expression.
- If the publication does not contain an `mcd` folder, the payload is assumed to be XMLNSC.

Read the following steps to see how to enable content-based filtering on IBM App Connect Enterprise.

Procedure

1. Set the `cbfEnabled` property of the `ContentBasedFiltering` object for the integration server in which you want content-based filtering to run.

   Note that you must explicitly enable the `cbfEnabled` property for content based filtering to work; the default setting is for content based filtering to be off.
2. Restart the integration server for the change to take effect.

   If you enable content based filtering in multiple integration servers on z/OS, content based filtering is active in only one integration server at any time. Subsequent integration servers for which content based filtering is enabled, propagate the following messages to the syslog on start up (for each content based filtering thread) and then every 30 minutes, as they fail to connect to the queue manager:

   ```
   BIP2111E MQ04BRK jheg1 15 IBM App Connect Enterprise INTERNAL ERROR: DIAGNOSTIC INFORMATION
    'Error occurred in Content Based Filtering Thread'. : ImbCbfWorker(909)
   BIP2624E MQ04BRK jheg1 14 UNABLE TO CONNECT TO QUEUE MANAGER 'MQ04': MQCC=2;
    MQRC=2002; MESSAGE FLOW NODE 'ContentBasedFiltering'. : ImbCbfWorker(214)
   ```

   If you stop the integration server that is currently providing content based filtering services, another integration server for which content based filtering is enabled connects to the queue manager and provides content based filtering services.

   Both the `evaluationThreads` and `validationThreads` properties default to one if content-based filtering is enabled.

   Evaluation threads are used to validate content filters against a given publication at publication time. A network with a high number of publications might require the `evaluationThreads` property to be increased (up to a maximum of 32) to handle the workload at publication time.

   Validation threads are used to validate syntax of content filters at subscription time. A publish/subscribe network with a high number of subscribers, especially if dynamic subscribers, might require the `validationThreads` property to be increased (up to a maximum of 32) to handle the high throughput of subscription requests.

   It is possible to enable this function in multiple integration servers, but you must ensure that any message sets required to parse any published message (and referenced in the `mcd` folder of that message) are deployed to all the integration servers that have been enabled for content-based filtering.

   Any errors encountered parsing the message within the evaluation thread cause IBM MQ to return MQRC_CONTENT_ERROR to the publishing application. The parsing error appears as well in the event log as an IBM App Connect Enterprise exception.

   Example of a subscription <psc> folder processing ESQL and message properties in the filter:

   ```
   <psc>
   <Topic>topic</Topic>
      <Filter>
      SUBSTRING(Root.XMLNSC.Name.FirstName FROM 1 FOR 1) = ‘J' and usr.flag = ‘yes'
      </Filter>
   </psc>
   ```

Using Operational Decision Manager (ODM) business rules

You can configure message flows in IBM App Connect Enterprise to execute Operational Decision Manager (ODM) business rules, by using either an ODMRules node or a JavaCompute node.

Business rules are policies, constraints, or required operations that apply to a specific set of business conditions or dependencies. For example, a bank could use a business rule to decide whether a credit check is required for a customer opening a new account; the rule could stipulate that a credit check is required for new customers, but not for existing ones.

When working with IBM Operational Decision Manager, business rules are organized in rulesets and RuleApps. A ruleset is a decision-making program that can be processed by a rule engine to yield a

decision, based on input and output parameters. The decision-making logic is implemented as rules, decision tables, and ruleflows. A RuleApp is a deployable management unit for ODM Rule Execution Servers and can contain one or more rulesets.

You can use an node in a message flow to execute a set of ODM business rules contained in a ruleset. An ODMRules node can be used to execute rulesets that have XML parameters. You must attach an ODM Server policy to an ODMRules node to control how it accesses a remote ODM Rule Execution Server; for more information, see ODM Server policy (ODMServer).

Alternatively, you can use a node to execute ODM business rules. A JavaCompute can be used to execute rulesets that have either Java or XML parameters. You must attach an ODM Server policy to a JavaCompute node to control how it accesses a remote ODM Rule Execution Server; for more information, see ODM Server policy (ODMServer). You can also use the Process via Operational Decision Manager ruleset wizard to create template Java code that connects to an Operational Decision Manager (ODM) Server.

Rulesets that are used by an integration server can be updated either automatically or manually when they are changed in the ODM Rule Execution Server (RES). You can update the rulesets automatically by configuring properties in an ODMServer policy, as described in "Configuring automatic ruleset updates" on page 1248, or manually by using the administration REST API, as described in "Configuring manual ruleset updates" on page 1249. This update mechanism applies to rulesets that are loaded from the ODM Rule Execution Server and to rulesets that have been downloaded to the local file system.

When a ruleset is loaded for the first time (whether through an ODMRules node or a JavaCompute node), it must be compiled by the ruleset engine before it can be used. This compilation can take several seconds, particularly for rulesets that contain XML parameters. By default, this compilation takes place when the first message attempts to execute the ruleset, but you can choose to compile the ruleset when it is loaded during flow startup, by setting the **`Ruleset load strategy`** property in the ODMServer policy. For more information, see ODM Server policy (ODMServer).

For more information about using ODM business rules, see the following topics:

- "Executing ODM business rules by using an ODMRules node" on page 1242
- "Configuring the ruleset path" on page 1243
- "Using a local ruleset archive" on page 1244
- "Executing ODM business rules by using a JavaCompute node" on page 1245
- "Creating ODM classes by using the Process via Operational Decision Manager ruleset wizard" on page 1247
- "Discovering and applying ruleset updates" on page 1248

Executing ODM business rules by using an ODMRules node

You can use an ODMRules node to execute IBM Operational Decision Manager (ODM) business rules.

Before you begin

Read "Using Operational Decision Manager (ODM) business rules" on page 1241.

About this task

You can use an ODMRules node to execute business rules contained in a ruleset, by completing the following steps:

Procedure

1. Create a message flow containing an node. For more information, see "Creating a message flow" on page 574.

2. Create an ODM Server policy that contains details of either a remote Operational Decision Manager (ODM) Rule Execution Server or a ruleset archive that has been downloaded and stored in the local file system.

   For more information, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327 and "Using a local ruleset archive" on page 1244.
3. Configure the ODMRules node to use the ODM Server policy that you created in step 2, either by specifying the policy name in the **Policy** field (in the form *policyProject*}:*policyName*), or by clicking **Browse** and selecting the required policy from the displayed list.
4. Configure the ruleset path in the ODMRules node, as described in "Configuring the ruleset path" on page 1243.

   When the ruleset path configuration wizard has completed, it populates the parameters table.
5. Configure a **Data location** XPath for each of the parameters listed in the table. Validation of the node cannot complete until this step has been completed.

   You can either enter the XPath manually or click the ellipsis (**...**) button to open the standard XPath editor, as described in node. For each parameter, the location is identified in the message tree. For IN and INOUT parameters, the location specifies where the required parameter data can be found in the input message tree, and that part of the message tree is serialized as XML. The resulting XML is then set as the parameter value when the ruleset is evaluated. For the IN and INOUT parameters, the location must exist so that it can be serialized. For the INOUT and OUT parameters, after executing the ruleset, the XPath location is used to inflate the XML that is obtained from the ruleset execution into the message tree.
6. When the **Data location** for has been configured for each parameter, save the changes.

Configuring the ruleset path

You can use the **Ruleset path** property of the ODMRules node to define the set of Operational Decision Manager (ODM) business rules that are to be evaluated for message processing.

Before you begin

Read the following topics:

- "Using Operational Decision Manager (ODM) business rules" on page 1241
- "Executing ODM business rules by using an ODMRules node" on page 1242

About this task

The set of rules to be evaluated for message processing is defined by the **Ruleset path** property of the node.

Procedure

Configure the ruleset path by completing the following steps:

1. On the **Basic** tab of the ODMRules node, click **Configure** next to the **Ruleset path** field.

   You can then use the interactive panels to choose a ruleset from those that are already deployed to a Rule Execution Server; you cannot edit the field manually.

   The value of the field follows the Operational Decision Manager (ODM) syntax: `'/RuleApp/RuleAppVersion/ruleset/rulesetVersion'`. The RuleAppVersion and RulesetVersion can reference either a specific version or the `latest` version. For example:

   ```
   '/SaleApp/1.1/SaleSet/2.0'
   ```

   ```
   '/SaleApp/latest/SaleSet/latest'
   ```

   ```
   '/SaleApp/2.2/SaleSet/latest'
   ```

2. When the Rule Set wizard opens, set the following properties in the **Connect to an ODM Server** panel to define access to the Rule Execution Server:
   a) In the **`URL of the ODM Rule Execution Server`** field, specify the Rule Execution Server context root URL. For example: `http://localhost:9080/res/`.
   b) In the **`Username`** field, specify the username to be used to access the server.
   c) In the **`Password`** field, specify the password to be used to access the server.
3. Click **Next**.
   A query is sent to the Rule Execution Server, and the results of the query are used to populate the fields on the **Select a Ruleset for the ODMRules node** panel:

Table 41. Select a Ruleset for the ODMRules node

Field	Description
RuleApp	The RuleApp containing the ruleset to be used. The drop-down menu offers a choice of applications deployed on the server.
RuleApp version	The version number of the RuleApp. The drop-down menu contains the available versions and the string 'latest'.
Ruleset	The name of the ruleset to use from the rule application selected above. The drop-down menu offers a choice of rulesets in the application.
Ruleset version	The version number of the ruleset. The drop-down menu contains the available versions and the string 'latest'.
Selected ruleset parameters	The list of parameters of the ruleset with their respective names, types, and directions. The `Direction` can be IN, OUT, or INOUT, for input, output, and bi-directional parameters respectively. The `Kind` field is always XML for the ODMRules node.
Create/update the xsd(s) required for the ruleset	If selected, the Toolkit imports the XSD schemas describing the message formats that are used by the ruleset and makes them available in the project.

4. Click **Finish**.

Using a local ruleset archive

You can configure an ODM Server policy to specify a local ruleset archive instead of an Operational Decision Manager Rule Execution Server.

Before you begin

Read the following topics:

- “Using Operational Decision Manager (ODM) business rules” on page 1241
- “Executing ODM business rules by using an ODMRules node” on page 1242

About this task

You can download an archive of rulesets from the Operational Decision Manager Rule Execution Server (RES) and store it on a local file system. You can then configure an ODM Server policy to specify this locally-stored archive as the location for the ruleset to be used by the message flow rather than accessing a ruleset on the ODM server.

Procedure

Configure an ODM Server policy to specify a local archive by completing the following steps:

1. Download an archive of the required rulesets from the Operational Decision Manager Rule Execution Server and store it on your local file system.
2. Create and configure an ODM Server policy.

   For information about creating a policy, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.
3. Set the **Rule Execution Server URL** property of the ODM Server policy to specify the location of the downloaded archive file.
   For example:

   ```
   file:/Users/admin/ACE/rulesets/ruleApp_Sale_1.0.jar
   ```

   or

   ```
   file:///c:\Users\Administrator\Downloads\ruleApp_Sale_1.0.jar
   ```
4. Save and deploy the policy.

   The message flow containing the node functions in the same way as it would if the ODM Server policy was configured to connect to a Rule Execution Server, but it accesses the rulesets that are stored on the local file system instead of connecting to the Rule Execution server.

Executing ODM business rules by using a JavaCompute node

You can use a JavaCompute node to execute IBM Operational Decision Manager (ODM) business rules.

Before you begin

- Read "Using Operational Decision Manager (ODM) business rules" on page 1241.
- Create a message flow containing a node.

About this task

You can use a JavaCompute node to execute rules contained in a ruleset, by using an embedded rule engine.

Procedure

To use a JavaCompute node to execute rules contained in a ruleset, complete the following steps:

1. Obtain a connection to an ODM server
2. Obtain a ruleset
3. Execute the ruleset

Obtaining a connection to an ODM server

About this task

Obtain a connection to an ODM Rule Execution server during the `onSetup` method of a JavaCompute node, by calling the `getODMServer` method and supplying the name of an ODMServer policy:

```
String odmServerPolicyName = "{myPolicyProject}:myODMServerPolicy";
MbODMServer odmServer = getODMServer(odmServerPolicyName);
```

This method initializes a connection to the ODM Rule Execution server, which must be running when the message flow that references the JavaCompute node is deployed or started. The `getODMServer` method can be called only from within the `onSetup` method. MbODMServer objects must not be shared between JavaCompute nodes and must not be cached for longer than the lifetime of the current node. MbODMServer objects that are obtained in the `onSetup` method can be used in the `evaluate` method to look up rulesets.

Obtaining a ruleset

About this task

When an MbODMServer object has been retrieved, it can be used to retrieve a ruleset from the ODM server for execution when the message flow is running. To retrieve a ruleset, call the `getRuleset` method on the MbODMServer object and supply the fully qualified path to the ruleset, in the form `'/RuleAppName/RuleAppVersion/rulesetName/rulesetVersion'`. You can specify either a particular version or `latest` to use the most recent version. For example:

```
MbODMRuleset odmRuleset = odmServer.getRuleset("/myRuleApp/1.0/my_ruleset/1.0");
```

```
MbODMRuleset odmRuleset = odmServer.getRuleset("/myRuleApp/Latest/my_ruleset/latest");
```

```
MbODMRuleset odmRuleset = odmServer.getRuleset("/myRuleApp/1.0/my_ruleset/latest");
```

When it is first referenced, the `Latest` version of a ruleset or RuleApp is loaded and cached at the application level. As a result, all message flows and nodes inside the application have the same `Latest` version, regardless of when they are first referenced. The latest version is re-evaluated when the application is stopped and restarted.

The `getRuleset` method retrieves the ruleset from the ODM server and makes it available in the embedded rule engine, ready for execution.

Rulesets define ruleset parameters, which are used to pass data between the ruleset and the calling application. The ruleset can have parameters of type `Java` or XML, and rulesets that support both parameter types can be called from a JavaCompute node. If the ruleset has parameters of type `Java`, the classes that define the Java objects for those parameters must be made available to the JavaCompute node at development time and at runtime. If you use the JavaCompute node wizard to query the ODM server and generate template Java code, you also have the option of downloading the class files that define the parameters and make them available to the Java project that contains the JavaCompute node.

The `getRuleset` method can be called during the JavaCompute node's `onSetup` or `evaluate` methods, and if the same rule is requested multiple times, all calls after the first one return the version that has been downloaded and cached. MbODMRuleset objects must not be shared between JavaCompute nodes and must not be cached for longer than the lifetime of the current node. MbODMRulesets can be retrieved during a JavaCompute node's `onSetup` method and then used by its `evaluate` method, to avoid delaying the execution of the first message.

Executing a ruleset

About this task

When executing a ruleset, you must build up a map of parameters to supply to the `execute` method of the MbODMRuleset. The map must contain an entry for each parameter defined as an IN or INOUT parameter in the ruleset definition. For parameters of type XML, you must supply a Java string that contains an XML representation of the `xsd` type (simple or complex) defined for that parameter. For parameters of type `Java`, you must supply an instance of the Java class defined for that parameter. To execute the ruleset, call the `execute` method of the MbODMRuleset, passing in the map of parameters:

```
Map<String, Object> ruleInParameters = new HashMap<String, Object>();
MyObject myObj1 = new MyObject();
ruleInParameters.put("ruleParameter_Java_1", myObj1);
ruleInParameters.put("ruleParameter_XML_2", "<Customer><name>John Doe</name><age>33</age></
Customer>");
MbODMRulesetExecutionResponse ruleResponse = odmRuleset.execute(ruleInParameters);
```

The result of calling the `execute` method is an instance of an MbODMRulesetExecutionResponse. This object gives access to the map of OUT and INOUT parameters of the ruleset through the `getOutputParameters` method, and to some execution statistics:

```
Map<String, Object> ruleOutParameters = ruleResponse.getOutputParameters();
MyResultObject myResult1 = (MyResultObject)ruleOutParameters.get("ruleResponseParameter1");
long rulesFired = ruleResponse.getTotalRulesFired();
```

You can use normal message tree access and modification techniques to retrieve data from the incoming message tree to build the parameters supplied to the ruleset `execute` method, and to build or modify the outgoing message tree with details from the result parameters. The `execute` method of the MbODMRuleset can be called multithreaded, and it does not need to be looked up for each flow instance.

Executing rulesets by using a JavaCompute node

About this task

When you create a JavaCompute node, you are presented with a series of templates, one of which is **Process via Operational Decision Manager ruleset**. Select this option to query the available rulesets in an ODM Rule Execution server; the wizard guides you through selecting a ruleset and then generates template Java code with details about the rule and how to call it. For more information, see "Creating ODM classes by using the Process via Operational Decision Manager ruleset wizard" on page 1247.

Creating ODM classes by using the Process via Operational Decision Manager ruleset wizard

You can use the **Process via Operational Decision Manager ruleset** wizard within a JavaCompute node to create Java code that executes a ruleset retrieved from an Operational Decision Manager (ODM) Server.

Before you begin

- Read "Using Operational Decision Manager (ODM) business rules" on page 1241 for information about using ODM business rules with IBM App Connect Enterprise.
- Create a JavaCompute node in your message flow.

About this task

You can use the **Process via Operational Decision Manager ruleset** wizard to query the available rulesets in an ODM Rule Execution server, select a ruleset, and then generate Java code with details about the rule and how to call it.

When you create a JavaCompute node, templates are available to help you develop your Java code. Select **Process via Operational Decision Manager ruleset** to use the wizard.

For information about using a JavaCompute node to execute ODM business rules, see "Executing ODM business rules by using a JavaCompute node" on page 1245.

Alternatively, you can use an ODMRules node in a message flow to execute a set of business rules, as described in node.

Procedure

Complete the following steps to use the **Process via Operational Decision Manager ruleset** wizard to query the available rulesets in an ODM Rule Execution server, select a ruleset, and then generate Java code with details about the rule and how to call it:

1. In the IBM App Connect Enterprise Toolkit, double-click the JavaCompute node in your message flow and click **Process via Operational Decision Manager ruleset**.
2. Follow the instructions in the wizard.
3. Click **Finish**.

Discovering and applying ruleset updates

Rulesets that are used by an integration server can be refreshed either automatically or manually when they are changed in the Operational Decision Manager (ODM) Rule Execution Server.

You can discover and apply updates to ODM rulesets automatically, by configuring properties in an ODMServer policy, or manually, by using the administration REST API. IBM App Connect Enterprise queries the ODM Rule Execution Server for information about all rulesets that are currently in use in the integration server, either through an ODMRules node or a JavaCompute node. Checks are made to see whether either of the following conditions has been met:

- An existing ruleset has been updated.
- A new ruleset has been deployed that allows a `latest` ruleset reference to resolve to a later version (as described in "Configuring the ruleset path" on page 1243).

If either of these cases is true, the new ruleset is downloaded and used by the integration server.

When a new ruleset is loaded, the details are reported by the integration server through BIP messages that are sent to the event log:

- `bip4337. Ruleset '&1' from ODMServer policy '&2' has been updated.`
- `bip4338. Ruleset '&1' from ODMServer policy '&2' has been updated and now resolves to ruleset '&3'.`

If any problems occur when the updated ruleset is being downloaded or compiled, the issues are reported through BIP error messages.

To discover and apply ruleset updates, complete the steps in one of the following topics:

- "Configuring automatic ruleset updates" on page 1248
- "Configuring manual ruleset updates" on page 1249

Configuring automatic ruleset updates

You can configure IBM App Connect Enterprise to discover and apply updates to Operational Decision Manager (ODM) rulesets automatically, by setting properties in an ODMServer policy.

Before you begin

Read "Using Operational Decision Manager (ODM) business rules" on page 1241.

About this task

To discover and apply ruleset updates automatically, complete the following steps:

Procedure

1. Complete the steps in one of the following tasks, according to your chosen method:
   - "Executing ODM business rules by using an ODMRules node" on page 1242
   - "Executing ODM business rules by using a JavaCompute node" on page 1245
2. Set the following properties in the ODMServer policy that is specified by your ODMRules or JavaCompute node:
   a) Set the **`Ruleset refresh mode`** property to `polling`.
   b) Optional: Set the polling interval that you require by specifying a value (in seconds) in the **`Ruleset refresh polling interval`** property.
   The default value for this property is `600` seconds (10 minutes).
   For more information, see ODM Server policy.
3. Deploy the ODMServer policy. For more information about policies, see "Overriding properties at run time with policies" on page 324 and "Policies overview" on page 324.

4. Run the flows that use the ODMServer policy as usual.

   When a ruleset has been updated in the rule execution server or in the local file system, the updated ruleset is discovered and applied automatically when the polling interval expires. When the update has completed, a message is written to the syslog, and the changes to the ruleset are applied automatically to the flows that use it.

Configuring manual ruleset updates
You can use the administration REST API to discover and apply updates to Operational Decision Manager (ODM) rulesets.

Before you begin

Read "Using Operational Decision Manager (ODM) business rules" on page 1241.

About this task

To discover and apply ruleset updates manually, complete the following steps:

Procedure

1. Complete the steps in one of the following tasks, according to your chosen method:
   - "Executing ODM business rules by using an ODMRules node" on page 1242
   - "Executing ODM business rules by using a JavaCompute node" on page 1245
2. Use the administration REST API to discover and apply ruleset updates manually, by making an HTTP POST call to the appropriate URL:
   - Use the following URL to refresh all policies:

     ```
     http://host:4414/apiv2/resource-managers/odm/refresh-ruleset-cache
     ```

   - Use the following URL to refresh a specific set of policies:

     ```
     http://host:4414/apiv2/resource-managers/odm/refresh-ruleset-cache?
     policy={myProject}:myPolicy&policy={myOtherProject}:myOtherPolicy
     ```

   - Use the following URL to refresh policies in a single project:

     ```
     http://host:4414/apiv2/resource-managers/odm/refresh-ruleset-cache?
     project=myProject&policy=myPolicy&policy=myOtherPolicy
     ```

3. The REST call starts a ruleset refresh in the background, and a message confirms that the update request has been started. The ruleset update is therefore asynchronous from the REST call. After the request has been made, messages are written to the syslog when the rulesets are updated.

   For more information about using the administration REST API, see "Managing resources by using the administration REST API" on page 334.

Transforming and enriching messages

Transform and enrich messages by using one or more of the following techniques.

About this task

Use one or more of the following options for transforming and enriching the messages in your message flows.

Visual transformation
Use the Mapping node to transform the incoming message, create new output messages, and interact with information in a database by using a graphical data map.

Stylesheet
Use the XSLTransform node to transform the incoming XML message by using an XSL stylesheet.

Programming languages
Use the following nodes to route and transform the incoming message, and create new output messages, by using programming languages:

- .NETCompute node:
  - Uses the .NET C#, VB or F# programming languages.
  - Can also interact with .NET modules.
- JavaCompute node:
  - Uses Java.
  - Can also use JAXB to model message data.
- Compute node:
  - Uses the ESQL language.

For more information about these options, see "Client application programming interfaces" on page 499. To use some of these options for transforming and enriching messages, you might need to create a model of the messages that you want to transform. For information about how to develop a message model, and why you might want to develop a message model, see "Constructing message models" on page 2133.

For details of the tasks that are associated with these options, use the instructions in the following sections:

- "Data Analysis" on page 1250
- Using message maps
- "Developing ESQL" on page 1607
- "Developing Java" on page 1751
- "Using XSL Transform" on page 1789
- "Using .NET" on page 1790
- "Combining a result message with an input message when fetching data from external systems" on page 1811

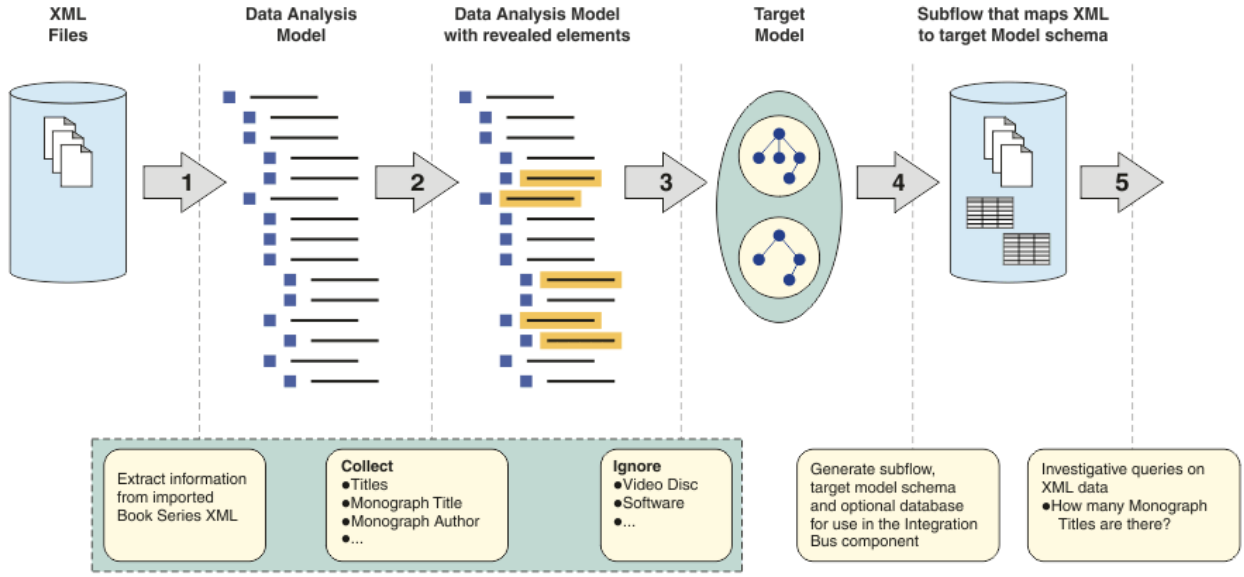
Data Analysis

Data Analysis analyzes and filters information in complex XML documents. You can use this analysis to create a library, that contains Data Analysis tools to quickly and easily transform your data in IBM App Connect Enterprise.

XML files are flexible, but can be long and complex. You might be interested in a small subset of data. However, it can be difficult to construct a model to filter and analyze this subset of data using traditional methods. Existing business intelligence (BI) tools work best with well-structured relational data.

In a Data Analysis project, you analyze a set of sample XML documents according to the content of the data. This content is defined in a Data Analysis profile that you specify when you create the project. When you analyze your sample XML documents, new data and new data structures within them are retained in the Data Analysis model. Repeated data and data structures are not added. After this analysis is complete, you can explore your Data Analysis model, which is a summary of your sample data, select relevant elements in the Data Analysis model, and then add them to a target model. Use the target model to produce a library that contains your Data Analysis tools, which you can use at run time to transform incoming data. The Data Analysis tools that are created include a map, schema file, validation stylesheet and subflow.

An overview of the process is shown in the following figure:



The key steps are as follows:

1. Create a Data Analysis project. See “Creating a Data Analysis project” on page 1251.
2. Analyze sample XML documents. See “Analyzing sample XML documents” on page 1252.
3. Create a target model. See “Creating a target model” on page 1255.
4. Populate and edit your target model. See “Populating a target model from your sample XML documents” on page 1255 and “Modifying a target model in the Target Model editor” on page 1256.
5. View and edit your target model in database format. See “Editing the database representation of the target model in the Target Model editor” on page 1260 and “Data Analysis database setup” on page 1259.
6. Create a library that contains Data Analysis tools. See “Creating Data Analysis tools” on page 1258.
7. Use the Data Analysis tools to work with your data in the Integration Development perspective.

Creating a Data Analysis project

Create a Data Analysis project to create, contain, and develop your Data Analysis model, target models, and Data Analysis tools.

Procedure

1. In the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Data Analysis Project**.
   The **New Data Analysis Project** wizard opens.
2. Complete the fields in the **New Data Analysis Project** wizard and click **Finish**.
   The Data Analysis perspective and the Data Analysis Project Overview open.

Results

Your Data Analysis project is created and the Data Analysis Project Overview is open in the editor area.

What to do next

Create a Data Analysis model by analyzing sample XML documents. See “Analyzing sample XML documents” on page 1252.

Analyzing sample XML documents

Use the Data Analysis Project Overview to analyze sample XML documents, and to add data structures that conform to your schema into a Data Analysis model. Select elements and attributes from your Data Analysis model to form a target model, which you use to create a library that contains Data Analysis tools.

Before you begin

- Create a Data Analysis project. See "Creating a Data Analysis project" on page 1251.
- Ensure that your sample documents are in XML format. The sample XML documents must be representative of your relevant data so that you can later produce a useful target model. If data is missing from your sample XML documents, subsequent input may not conform to the generated target model.
- Ensure that the sample XML documents conform to a schema and that the schema is available. If sample data does not conform to your schema, you cannot analyze your sample XML documents, nor load the relevant data from within them.

Procedure

1. In the Data Analysis Project Overview, click **Analyze documents**.

   The **Analyze documents** wizard opens.
2. Complete the steps in the **Analyze documents** wizard and click **OK**.

   The **Summary of Analyzed Documents** window opens, and shows whether you successfully analyzed your documents.

   Note:

   - At most, the analysis adds 10 identical data items or data structures from each document. When you analyze additional documents, pre-existing data or data structures are ignored and therefore are not displayed in any of the views.
   - The time that is taken to analyze the sample XML documents increases with file size.
3. Click **OK** to close the **Summary of Analyzed Documents** window.
4. Optional: Repeat these steps to analyze additional sample documents.

Results

Your sample XML documents are analyzed, ready for you to create a target model.

What to do next

You can now complete the following task:

- Create a target model. See "Creating a target model" on page 1255.

Exploring data in the Data Analysis Model view

Use the Data Analysis Model view to explore your analyzed data.

Click the following headings to learn how to use the Data Analysis Model view to explore your data:

- "Sorting elements" on page 1253
- "Highlighting elements" on page 1253
- "Choosing which element to display" on page 1254
- "Displaying data from a selected element" on page 1254
- "Viewing document analysis history" on page 1254

Sorting elements

By default, the elements are displayed in the order in which they are added from the analysis of your sample XML documents. Click the **Display the tree in an alphabetical order** icon ( ) at the top of the view to view the elements in alphabetical order. Click this icon again to return the list to the original order.

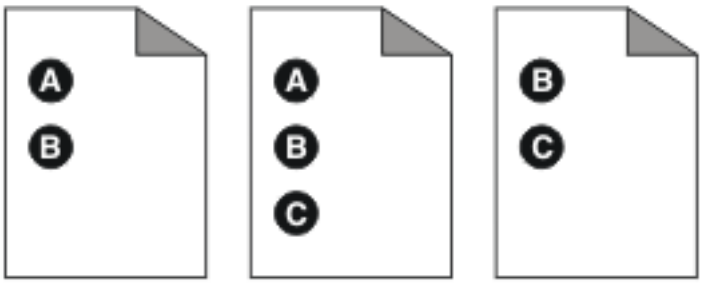
Highlighting elements

Highlight elements in the Data Analysis Model view to identify elements that have common features. Right-click in the Data Analysis Model view and then select one of the following options:

Highlight All Coexisting Elements

Use this option to highlight all elements that are in one or more of the same sample XML documents as your selected element. This action shows where combinations of data structures occur. The selected element is highlighted in the Data Analysis Model view, and is also identified at the bottom of the view, as the *Focus Element*. The percentage next to each element indicates the percentage of documents that contain both that element and the focus element.

This concept is illustrated by the following examples, in which each of three sample XML documents contain a combination of elements A, B, and C:



Example 1: Element A is the focus element.

Element B is in all of the sample XML documents that contain element A, so the percentage of coexistence for B is 100%.

Element C is in only one of the two sample XML documents that contain element A, so the percentage of coexistence is 50%.

These figures are displayed in the tree as B [100%] and C [50%].

Example 2: Element B is the focus element.

Element A is in two of the three sample XML documents that contain element B, so the percentage of coexistence is 66.6%.

Element C also has a percentage of coexistence is 66.6%, for the same reason.

Highlight Elements Based on Sample Documents

Use this option to highlight elements that were added to the Data Analysis model during the analysis of your sample XML documents. Use the **Sample Documents** window to select the sample XML documents. The elements that were added from the selected documents are highlighted.

For example, if you select documents 'exampleDoc1' and 'exampleDoc5' in the **Sample Documents** window, the elements that are highlighted are those that were added when documents 'exampleDoc1' and 'exampleDoc5' were analyzed.

Highlight Elements Based on Time

Use this option to highlight elements that were added or modified after a certain time, which you specify. You can use one of the following options to determine how the elements are highlighted:

- **Added After**
- **Added or Modified After**

- **Modified After**

Tip: Click **Off** in the lower left corner of the Data Analysis Model view to turn off any highlighting.

Choosing which element to display

View a greater or smaller number of elements to help provide context or focus. Use one of the following options to change how many elements are displayed:

Expand All Descendants
Display elements that are nested under other elements. Display descendant elements by clicking **Expand All Descendants** on the Data Analysis Model view menu, or click the **Expand All Descendants** icon () at the top of the view.

Collapse All Descendants
Hide elements that are nested under other element to focus on parent elements. This action reduces the amount of displayed data. Collapse descendant elements by clicking **Collapse All Descendants** on the Data Analysis Model view menu, or click the **Collapse all descendants** icon () at the top of the view.

Show Only Coexisting Elements
Show only elements that are in the same documents as the focus element. This action shows where data structure combinations occur. The focus element is highlighted in the tree and is also identified at the bottom of the view. When you highlight all coexisting elements, a percentage is displayed next to individual elements. This percentage indicates the percentage of documents that contain both the displayed element and the focus element. See "Highlighting elements" on page 1253 for an explanation of coexisting elements.

Displaying data from a selected element

To display data from a specific element, select the element and click **Show Data Under the Selected Element** from the Data Analysis Model view menu. This action prevents excess data from being displayed and reduces the time that you spend searching for data. The data from the selected element is displayed in the Data Filter view.

Tip: Click **Off** in the Data Filter view to turn off this option.

Viewing document analysis history

Use the **Show Document Analysis History** option in the Data Analysis Model view menu to display the following information about each time that you analyze your sample XML document:

- Start time
- Completion time
- Total amount of documents examined
- A list of documents that were successfully analyzed
- The number of documents that were successfully analyzed
- The number of schema validation errors
- The number of XML parsing errors
- The number of elements that were added to the Data Analysis model

Next:

- Explore your data further.
- Add to your target model. See "Populating a target model from your sample XML documents" on page 1255.
- Create a library that contains your Data Analysis tools. See "Creating Data Analysis tools" on page 1258.

Creating a target model

Create a target model, which is based on the data from your analyzed sample XML documents. Use your target model to create a library that contains Data Analysis tools to transform data.

Before you begin

- Create a Data Analysis project. See “Creating a Data Analysis project” on page 1251.
- Analyze your sample XML documents. See “Analyzing sample XML documents” on page 1252.

Procedure

1. In the Data Analysis Project Overview, click **Create a new target model**.
   The **Confirm** window opens.
2. In the **Confirm** window, click **OK** to save your Data Analysis project.
   The **New Target Model** wizard opens.
3. Complete the fields in the **Target Model** wizard and click **Finish**.

Results

Your target model is created and the Target Model editor opens in the Editor area.

What to do next

- Populate your target model. See “Populating a target model from your sample XML documents” on page 1255.
- Add more sample XML documents to the Data Analysis model. See “Analyzing sample XML documents” on page 1252.
- Create your library that contains Data Analysis tools. See “Creating Data Analysis tools” on page 1258.

Populating a target model from your sample XML documents

Populate your target model, so that you can use it to create a library that contains your Data Analysis tools.

Before you begin

Before your start:

- Create a Data Analysis project. See “Creating a Data Analysis project” on page 1251.
- Analyze sample XML documents. See “Analyzing sample XML documents” on page 1252.
- Create a target model. See “Creating a target model” on page 1255.

About this task

You can populate your target model from the Data Filter view, the Data Paths view, and the Data Analysis Model view.

To populate your target model, complete the following steps:

Procedure

1. In the Data Filter view, search for data structures by typing in the **Filter (by Element Name or Element Data)** field.
   The search results are displayed in the table.
2. In the Data Filter view, click an element to display a list of instances where that data structure is contained in your sample XML documents.
   This list is displayed in the Data Paths view.

3. Drag elements from the Data Analysis Model view, Data Filter view or, Data Paths view to the Target Model editor to populate the target model. If you drag an element from the Data Filter view that has multiple possible data paths, a **Data Paths** window opens. Select a data path from the list and click **Finish**.

Results

Your target model is populated.

What to do next

- Edit your target model. See "Modifying a target model in the Target Model editor" on page 1256.
- Create your Data Analysis tools. See "Creating Data Analysis tools" on page 1258.

Modifying a target model in the Target Model editor

You can use the Target Model editor to alter the structure of your target model to meet the requirements of the downstream application.

Before you begin

- Create a target model. See "Creating a target model" on page 1255.
- Populate your target model. See "Populating a target model from your sample XML documents" on page 1255.

About this task

Double-click the following headings to learn how to modify your target model with the Target Model editor:

- "Creating local elements" on page 1256
- "Renaming elements and attributes" on page 1256
- "Moving elements" on page 1257
- "Deleting elements and attributes" on page 1257
- "Restoring deleted elements and attributes" on page 1257

Creating local elements

About this task

Create local elements in your target model

Procedure

1. Right-click the Target Model editor and select **New Child Local Element** or **New Sibling Local Element**. The **New Element** window opens.
2. Type the name for your new element in the **Element name** field and click **OK**.

Results

Your new element is created.

Renaming elements and attributes

About this task

Rename elements or attributes so that your target model is compatible with other applications.

Procedure

1. In the Target Model editor, right-click the element or attribute that you want to rename, and select **Rename** from the menu.

   The **Rename** window opens.
2. In the **New name** field, type the new name for your element or attribute and click **OK**.

Results

Your element or attribute is renamed.

Moving elements

About this task

Move elements in the Target Model editor so that they are attached to different data structures. Change the order of sibling elements to affect the function of your Data Analysis tools.

Procedure

In the Target Model editor, drag the element or attribute from one position to another under its parent element.

Results

Your element or attribute is in a new position within the target model structure.

Deleting elements and attributes

About this task

Delete elements or attributes from your target model.

Procedure

- To delete an element or attribute, complete the following step:
  a) In the Target Model editor, right-click the element or attribute that you want to delete and select **Delete** from the menu.

  The element or attribute is deleted.
- To delete elements or attribute that have similar data paths to the one that is in focus:
  a) In the Target Model editor, right-click an element or attribute and select **Delete similar elements or attributes**.

     The **Item selection for removal** window is displayed.
  b) Select the elements or attributes that you want to delete from the list and click **Ok**. The element or attribute that is in focus is also displayed on this list.

  The selected elements or attributes are deleted.

Restoring deleted elements and attributes

About this task

Restore a deleted element or attribute to your target model.

Procedure

- To restore a deleted element, drag it from the Data Analysis Model view to the Target Model editor.
- To restore a deleted attribute, select **Restore Deleted Attributes** from the context menu in the Target Model editor.

Results

Your element or attribute is restored.

What to do next

- Use your target model to create Data Analysis tools. See “Creating Data Analysis tools” on page 1258.
- Prepare your database to use with the Data Analysis perspective. See “Data Analysis database setup” on page 1259.
- Use the Target Model editor to work with the data from your database. “Editing the database representation of the target model in the Target Model editor” on page 1260.

Creating Data Analysis tools

Use your target model to create a library. This library contains Data Analysis tools, which you use to transform your data.

Before you begin

- Analyze your sample XML documents. See “Analyzing sample XML documents” on page 1252.
- Create a target model. See “Creating a target model” on page 1255.
- Populate your target model. See “Populating a target model from your sample XML documents” on page 1255.
- Modify your target model. See “Modifying a target model in the Target Model editor” on page 1256.

About this task

You can use your target model to create a library, which contains the following Data Analysis tools:

- Schemas
- Maps
- Subflows
- Validation style sheets

Procedure

1. In the Target Model editor, click **Generate** to open the **Generate Data Analysis Tools** wizard.
2. Optional: In the **Generate Data Analysis Tool** wizard, select **Insert transformed messages into database**.
3. Complete the steps in the wizard and click **Finish**.

Results

A library that contains your Data Analysis tools is created. Your subflow opens in the editor.

What to do next

- Open the Application Development view and use your Data Analysis tools to work with your data.
- Work with the database. See “Editing the database representation of the target model in the Target Model editor” on page 1260 and “Data Analysis database setup” on page 1259.

Data Analysis database setup

Set up your database so that you can use it with Data Analysis.

Before you begin

- Create a database in one of the supported databases. The following databases are supported:
  - DB2
  - Oracle
  - Microsoft SQL Server

  Note: Remember to set a large enough page size to view the column lengths in the target model. If the page size is too small, an error message is produced in the database during table creation.
- Create a Data Analysis project. See "Creating a Data Analysis project" on page 1251.
- Analyze your sample data. See "Analyzing sample XML documents" on page 1252.
- Create your target model. See "Creating a target model" on page 1255.
- Generate your Data Analysis tools. See "Creating Data Analysis tools" on page 1258.

Procedure

1. In the Integration Development perspective, generate the DDL script:
   a) In the Project Explorer view, expand the project folder.
   b) In the Data Models folder, expand the `.dbm` file.
   c) Right-click the database name and select **Generate DDL**.
   d) To configure the DDL script, follow the steps in the **Generate DDL** wizard and click **Finish**.
   - Your SQL script is generated. This script shows the commands that are used to create your database tables.
   - The 'DOCID' column is added to your database. At runtime this column is populated with a unique string, to provide a map with a unique input key for database operations. Alternatively, you can set the input key yourself, by setting

     ```
     LocalEnvironment.Database.Input.Key
     ```

     in your local environment.
   - The 'ID' column is added to your database. The 'ID' is the foreign key that links related tables together.

   Tip: If you do not have a `.dbm` file, open the **Generate Data Analysis Tools** wizard and check the **Insert transformed messages into database** box.
2. If you are configuring a Microsoft SQL Server 2008, above the `Create Schema` line in your SQL script insert `DROP SCHEMA "SCHEMA_NAME" GO`. For example:

   ```
   DROP SCHEMA "MYSCHEMA"
    GO

    CREATE SCHEMA "MYSCHEMA"
    GO

    CREATE TABLE "MYSCHEMA"."MYTABLE" (
     "COLUMN1" VARCHAR(1000) NOT NULL,
     "COLUMN2" INT NULL
    )
    GO
   ```
3. If you are using Oracle, ensure that your schema name is the same as your username.
4. In your database configuration manager, run the SQL script.
5. In IBM MQ, create a JDBC connection so that the schema that is specified in the map is used.

Editing the database representation of the target model in the Target Model editor

Use the Target Model editor to view and edit your database representation, which is generated from your target model.

Before you begin

- Create your target model. See "Creating a target model" on page 1255.
- Populate your target model. See "Populating a target model from your sample XML documents" on page 1255.
- Set up your database. See "Data Analysis database setup" on page 1259.

About this task

See the following topics to learn how to use different features in the Target Model editor:

- "Viewing your database representation" on page 1260
- "Renaming tables and columns" on page 1260
- "Assigning different data types to elements" on page 1261
- "Deleting cell contents" on page 1261
- "Highlighting elements or attributes that are in the same table" on page 1261

Viewing your database representation

About this task

View your target model in a database format. You can later use IBM App Connect Enterprise to insert information into an existing database.

Procedure

1. Click the **Enable relational database mapping** icon ().
   The **Destination Database Platform Selection** wizard opens.
2. Complete the **Destination Database**, and **Destination Schema** fields, and select the product and version of your database. Click **OK**.
   Blank Table, Column, and SQL Data Type columns are displayed. The **Relational mapping options** icon (), at the upper right of the Target Model editor, is also in focus.
3. To generate the default SQL table naming for your target database, click the **Relational mapping options icon** () and select **Generate Default SQL Table Naming**.

Results

The Target Model editor is populated with the details of your target model, in a database format.

Renaming tables and columns

About this task

Rename tables or columns to change the location of an element within your database representation.

Procedure

1. Double-click the table or column cell that you want to alter.
   The cell is highlighted for editing.
2. Type the new name of the table or column.

Results
The selected table or column is renamed.

Assigning different data types to elements

About this task
Change the SQL Data Type of elements to change the type of data the elements can hold.

Procedure
1. Double-click the SQL Data Type cell that you want to alter.
   An ellipsis button is displayed in the cell.
2. Click the ellipsis button to display the **SQL Data Type** window.
3. Complete the fields and click **OK**.

Results
The SQL Data Type is changed.

Deleting cell contents

About this task
Delete the contents of cells in your database representation.

Procedure
- To delete the contents of individual cells, complete the following steps:
  a) Double-click the cell to highlight it for editing.
  b) Press the Delete key.
- To delete the contents of a row, but not the element name, complete the following steps:
  a) Right-click a cell in the row that you want to delete the contents of.
  b) Select **Reset SQL Naming** from the menu.
- To delete the contents of all of the cells, except for the element names, complete the following steps:
  a) Click the **Relational mapping options icon** () at the upper-right of the target model editor.
  b) Select **Reset SQL Naming** from the menu.
  The contents of cells are deleted.

Highlighting elements or attributes that are in the same table

About this task
Highlight elements or attributes that are in the same table so that it is easier to view the contents of a specific table.

Procedure
1. Right-click a cell in the table that you want to highlight.
2. Select **Show Elements or Attributes With the Same Table**

Results
Elements and attributes that are from the same table are highlighted.

What to do next

Create your Data Analysis tools. See “Creating Data Analysis tools” on page 1258.

Moving target models between Data Analysis projects

You can move similar target models from one Data Analysis project to another.

Before you begin

- Create and populate a target model. See “Creating a target model” on page 1255 and “Populating a target model from your sample XML documents” on page 1255.

  Note: Save your target model before you move it to another Data Analysis project.
- Create an additional Data Analysis project. See “Creating a Data Analysis project” on page 1251.

About this task

You might want to use a target model in another Data Analysis project than the one in which it was created. Complete the following steps to move your target model to an alternative Data Analysis project:

Procedure

1. In the Application Development view, copy and paste your target model to the alternative Data Analysis project.
2. Double-click your target model.
   - If the target model is not compatible with the Data Analysis project that you moved it to, the **Compatibility Check Result** window opens.
   - If the target model is compatible, the **Link Model** window opens.

   Important: To make an incompatible target model compatible with the new Data Analysis project, delete some target model elements in the **Compatibility Check Result** window. The more incompatible a target model is, the more elements are deleted. If a target model has no elements that are compatible with the new target model, then all of its elements are deleted.
3. Complete the steps in the **Compatibility Check Result** or **Link Model** window.

   Your target model is linked to the alternative Data Analysis project.

Results

The relocated target model opens in the Target Model editor.

What to do next

- Modify the target model. See “Modifying a target model in the Target Model editor” on page 1256.
- Create Data Analysis tools. See “Creating Data Analysis tools” on page 1258.
- Create or move new target models. See “Creating a target model” on page 1255.

Glossary File Lookup Service

Glossary File Lookup Service uses glossary files to transform the data element code in your sample XML documents into more readable display names. These display names are viewed in your Data Analysis project.

Data element code in your sample XML documents can be difficult to understand, because it often consist of strings of numbers rather than descriptive words. Glossary files contain codes for each data element and their equivalent display names. The Glossary File Lookup Service extracts the display names, and shows them instead of the data element codes in your Data Analysis project. The Glossary File Lookup Service makes it easier to use Data Analysis because it is simpler to understand display names than their equivalent data element codes.

The Glossary File Lookup Service is in **Window** > **Preferences** > **Integration Development** > **Data Analysis**.

Using message maps

You can use a message map to graphically transform an input message into a required output message; to enrich the output message with data from a database; to dynamically set routing or destination control for the output message; and to modify data in a database system. You can use drag actions to make connections, select transforms, and build logic to transform your message data without programming.

About this task

A message map is the IBM App Connect Enterprise implementation of a graphical data map. It is based on XML schema and XPath 2.0 standards, with additional support for JSON schema draft 4.

A message map offers the ability to achieve the transformation of a message without the need to write code. It provides a visual image of the transformation, and simplifies its implementation and ongoing maintenance.

You can use a message map to adopt any of the following integration requirements graphically:

- **Transform a message**: You can use a message map to graphically transform a message assembly, message body, and properties, according to the transforms and XPath functions defined in the message map. You can use the full set of XPath 2.0 expressions and functions to implement data calculations and manipulations in a message map. To define the input and output messages to a map, you can use a schema base message model, which defines the structure of the data and provides information about the data type, or you can define it dynamically in the map by using the **Add user defined** function.
- **Enrich a message with data available in an external database**: You can use a message map to enrich, or conditionally set the output message with data from a database table. The table data structure must be defined to the message map, and an SQL `where` clause can be used to select specific rows. The resulting row data is presented as an extra input in the message map, according to the database schema.
- **Modify data located in an external database**.
- **Route a message based on content**: You can use a message map to graphically route a message. You can modify the local environment tree to set a dynamic message destination.

Procedure

Read the following sections to learn how to design, create, configure, and troubleshoot a message map and its associated resources:

- Graphical data maps offer the ability to achieve the transformation of a message without the need to write code. Depending on the data transformation re-usability and manageability requirements, you can use a message map, a submap, or a local map. For more information, see “Graphical Mapping overview” on page 1264.
- You can create a graphical data map, a message map, or a submap to transform a message.

  For more information, see “Creating message maps” on page 1378.
- You can edit a message map by using the Graphical Data Mapping editor.

  For more information, see “Editing message maps” on page 1394.
- You can use the Graphical Data Mapping editor to set the value of an output element by using an expression, a transform, or a function.

  For more information, see “Setting the value of an output element by using a transform or a function” on page 1480.
- You can reference a message map during the development phase. You can also reference a message map dynamically at run time.

  For more information, see “Referencing message maps in your solution” on page 1537.

- You can diagnose and solve problems that you encounter when you use a message map.
  For more information, see “Troubleshooting a message map” on page 1587.
- IBM App Connect Enterprise does not support WebSphere Message Broker Version 7.0 legacy message maps (`.msgmap`). You can import message flows that contain legacy message maps into the IBM App Connect Enterprise Toolkit. However, you can view these message flows in read-only mode only, and you cannot deploy and run them. If you try to deploy a BAR file that contains a legacy message map, you see a BIP2355 error message. Before you run a message flow that contains a legacy message map or a legacy Mapping node, you must convert them by following the instructions in “Using or converting legacy resources into message maps” on page 1588.
- You can use the Graphical Data Mapping editor to access a user-defined policy.
  For more information, see “Accessing a user-defined policy from a graphical data map” on page 1370

Graphical Mapping overview

Graphical data maps offer the ability to achieve the transformation of a message without the need to write code. You can use a message map, a submap, or a local map. To define the input and output messages to a map, you can use an XML, JSON, or DFDL schema message model, which defines the structure of the data and provides information about the data type, or you can define it dynamically in the map by using the **Add user defined** function.

Message maps

A message map is the IBM App Connect Enterprise implementation of a graphical data map. It is based on XML schema and XPath 2.0 standards, with additional support for JSON schema draft 4 and Swagger 2.0.

A message map offers the ability to achieve the transformation of a message without the need to write code. It provides a visual image of the transformation, and simplifies its implementation and ongoing maintenance.

You can use a message map to graphically transform, route, and enrich a message. You can use a message map to modify data in a database system. You can use drag actions to make connections, select transforms, and build logic to transform your message data without programming.

For more information, see “Message maps” on page 1268.

Defining a map input and output

To enable graphical transformation, you must provide a model of the data so that the Graphical Data Mapping editor can display it. You can use an XML, JSON, or DFDL schema message model, or you can define a model dynamically in the map by using the **Add User-Defined** function. Predefined models are also provided for SOAP and BLOB message types. You can also transform data from other parts of the IBM App Connect Enterprise message assembly, including the Properties folder, the Environment, the Local Environment, and transport headers, such as HTTP or MQMD.

For more information, see “Message maps” on page 1268.

Submaps

A *submap* is a reusable form of message map.

You use a submap to reuse common data transformations. You define in the submap the mapping functions that transform a set of elements from the input object to the output object.

You can reuse submaps in other products that support graphical data maps.

Note: If you plan to reuse data transformations across different products, read “Guidelines for developing reusable graphical data mapping assets” on page 1273.

For more information, see “Submaps” on page 1271.

Local maps

A *local map* is a subset of data transformations between input elements and output elements that are part of a message map. You define a local map by creating a **Local map** transform in a message map.

A local map is not an independent resource. There is no physical file that is associated with a local map.

The scope of a local map is the message map. A local map is processed with the message map.

Local maps provide a way of breaking up a large message map into nested groups of mapping elements.

You can use local maps to simplify the overall message map presentation. You can structure complex data transformations into nested groups that are easier to manage and implement.

For more information, see "Local map" on page 1334.

Legacy message maps

A legacy message map is a message map that was created as a `.msgmap` file in earlier versions of WebSphere Message Broker (for example, in WebSphere Message Broker Version 7).

IBM App Connect Enterprise uses graphical data maps in `.map` format, which were introduced in WebSphere Message Broker Version 8. These message maps replace the previous message maps that were in `.msgmap` format.

Before you can use a legacy message map in IBM App Connect Enterprise, you must convert it into a message map, as described in "Using or converting legacy resources into message maps" on page 1588.

Choosing a type of graphical data map

Use the following table to identify the type of map that you must create when you transform data graphically in the Graphical Data Mapping editor:

Table 42. Types of map based on design requirements

	Recommended use	Type of resource	Supported in IBM App Connect Enterprise
Message map	Graphical data mapping	`.map` file	Yes
Submap (See notes below.)	Reuse of common data transformations	`.map` file	Yes
Local map	Reduce complexity when you read and manage a message map	No file. It is embedded within a Message map	Yes
Legacy message map	Reuse of maps that are developed in earlier versions of IBM App Connect Enterprise	`.msgmap` file	These files must be converted into `.map` files before they can be used in IBM App Connect Enterprise. (See notes below.)

Notes:

1. Legacy message maps must be converted into `.map` message map format before they can be deployed or modified by IBM App Connect Enterprise.
2. Submaps are not supported for use with JSON schema models.

Note:

Editing a graphical data map

You edit a message map or a submap in the Graphical Data Mapping editor.

The Graphical Data Mapping editor saves message maps as `.map` files.

For more information, see "Graphical Data Mapping editor" on page 1266 and "Editing message maps" on page 1394.

Mapping operations

You can use transforms to map graphically your data in the Graphical Data Mapping editor.

For more information, see "Transforms (Mapping operations)" on page 1280.

XPath

In the Graphical Data Mapping editor, you can use XPath functions in any of the following ways:

- You can define an XPath function to transform data by using a built-in XPath transform. For more information, see "Built-in XPath transforms" on page 1342.
- You can define complex XPath expressions that combine multiple XPath functions to transform data by using the **Custom XPath** transform. For more information, see "Custom XPath" on page 1315.
- You can define XPath expressions to set a condition on a transform or to filter an element in a repeating element. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.
- You can use XPath inline when you specify an argument to an XPath function. For more information, see "Built-in XPath transforms" on page 1342.

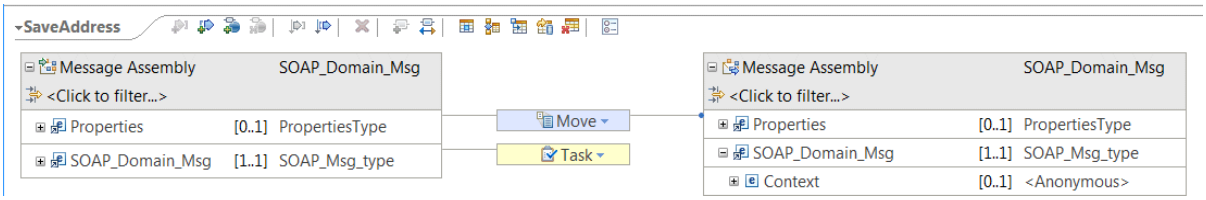
Built-in IBM App Connect Enterprise functions

You can use built-in IBM App Connect Enterprise functions to retrieve Integration Bus data and perform data-type conversion and formatting. The `iib:` functions are available through content assist in the "Custom XPath" on page 1315 transform. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282

Graphical Data Mapping editor
Use the Graphical Data Mapping editor to create and edit graphical data maps.

You can use the Graphical Data Mapping editor to take input (source) objects and transform them before you save the resulting output (target) objects.

Here is an example of the Graphical Data Mapping editor:



The input objects are shown on the left side of the canvas, and the output objects are shown on the right side. You can create connections between the input and output elements by clicking one element, and dragging the mouse to the element that you want to connect to. You can create connections either from left to right or from right to left. You can also right-click on either element and use the **Quick link** menu action to open a window where you can search for and select the other element.

You can use the Graphical Data Mapping editor to construct a graphical data map by using a wide variety of mapping transform functions. The transforms operate either from an input element to an output element on the canvas, or by directly setting the value of an output element.

You can use the functions either by using drag-and-drop, or from a menu action on the input or output element. The Graphical Data Mapping editor inserts the most appropriate mapping function on the newly created transform connection. For transforms that require multiple input connections, you can drag more connections onto the transform, and you can then select either a primary or supplementary connection mode. The Graphical Data Mapping editor adjusts the type of transform based on the new connections. When you create a connection, you can optionally change the type of transform function or start setting properties.

Some types of transforms involve complex input and outputs. They are edited by entering a nested view. The editor provides arrow buttons and breadcrumb navigation for nested transforms.

For information about the transforms that you can use in a graphical data map, see "Transform types in the Graphical Data Mapping editor" on page 1282.

The Graphical Data Mapping editor properties

The **Properties** pane displays the properties of the message map, its input and output elements, and transformations. For information about configuring the properties, see "Configuring the general properties of a message map" on page 1395.

When you are working with the maximized view of the Graphical Data Mapping editor, you can bring the **Properties** pane back into focus in either of the following ways:

- Right-click the object and then select **Show in** > **Properties view**.
- Use the appropriate keyboard combination, which, by default, is `Alt+Shift+Q, R`. You can configure these keyboard shortcuts by selecting **Window** > **Preferences** > **General** > **Keys**.

Actions supported in the Graphical Data Mapping editor

Table 43. List of actions supported in the Graphical Data Mapping editor

Icon	Label	Action
	**Add an input object**	Adds an input object to the map by selecting a map input.
	**Add an output object**	Adds an output object to the map by selecting a map output.
	**Add environment mapping**	Adds the Environment tree to the map.
	**Remove environment mapping**	Removes the Environment tree from the map.
	**Sort mapping by source**	Sorts the mapping objects so that the appearance of transforms is prioritized by the source.
	**Sort mapping by target**	Sorts the mapping object so that the appearance of transforms is prioritized by the target.
	**Delete selected elements**	Deletes the element or elements that are currently highlighted.
	**Create a new transform**	Creates a transform for the map.
	**Auto map input to output**	Automatically maps elements by using the **Automap Wizard**.

Table 43. List of actions supported in the Graphical Data Mapping editor (continued)		
Icon	**Label**	**Action**
	**Put key-value pair to cache**	Add a key-value pair to the global cache.
	**Get a key-value pair from cache**	Get a key-value pair from the global cache.
	**Remove a key-value pair from cache**	Remove a key-value pair from the global cache.
	**Select rows from a database**	Selects rows of data from a database table.
	**Call a database routine**	Calls a database routine to obtain the data to include in the map.
	**Insert a row into a database table**	Inserts one row into a database table.
	**Update rows in a database table**	Update one or more rows in a database table.
	**Delete rows from a database table**	Delete one or more rows in a database table.
	**Show object preferences**	Opens the **Preferences** window for the currently selected object.

Casting data structures

In the Graphical Data Mapping editor, you can use the **Cast** function to cast data structures.

Wildcards can be used to create a flexible message model that can be redefined when a more detailed definition is required. You define a wildcard as `xsd:any` in your schema.

When your message model schemas contain one or more wildcards, you can use the **Cast** function to redefine parts of the input or output model in a message map.

For more information, see "Casting elements in a message map" on page 1398.

Adding user-defined elements

In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to add dynamically user-defined elements that define an `xsd:any` element in your message model.

For more information, see "Adding and renaming a user-defined element" on page 1410.

Message maps
You can use a message map to graphically transform, route, and enrich a message. You can use a message map to modify data in a database system. You can use drag actions to make connections, select transforms, and build logic to transform your message data without programming.

A message map is the IBM App Connect Enterprise implementation of a graphical data map. It is based on XML schema and XPath 2.0 standards, with additional support for JSON schema draft 4 and Swagger 2.0.

You can use a message map to achieve the transformation of a message without the need to write code, providing a visual image of the transformation, and simplifying its implementation and ongoing maintenance.

The message map can be in the same container as the Mapping node (such as an application), or it can be in a referenced shared library. If multiple solutions are likely to use the same message map, store the map in a shared library.

Input and output data to a message map

In IBM App Connect Enterprise, the logical tree structure is the internal representation of a message. It is also known as the message assembly.

When a message arrives to IBM App Connect Enterprise, a parser is called. Each parser is suited for a particular type of message, and is known as a message domain. A parser converts the bit stream of an input message to its internal format. The data structure that you define in a message map for an input or an output message is the IBM App Connect Enterprise internal representation of the message.

To configure the input and the output to a message map, you define an input message assembly and one or more output message assemblies.

- You can have 1 input to a message map.
- You can have multiple outputs in a message map.

For more information, see "Input and output messages to a message map" on page 1276.

Supported message domains

The following message domains are supported when you use message maps:

- DFDL: You use this domain to manipulate general text or binary data streams including industry standards.
- XMLNSC: You use this domain to manipulate XML documents.
- SOAP: You use this domain to manipulate SOAP messages.
- DataObject: You use this domain to manipulate data without a stream representation.
- BLOB: You use this domain to manipulate messages with content that cannot be interpreted and subdivided into smaller sections of information.
- JSON: You use this domain to manipulate JSON documents.
- MRM: This domain is maintained for compatibility with earlier versions of IBM App Connect Enterprise.

Message assembly components

You can configure a message map to include the following message assembly components:

- The Properties folder.
- The message tree transport headers: For more information, see "Mapping transport headers" on page 1461.
- The local environment tree: For more information, see "Mapping data in the local environment tree" on page 1464.
- The message body
- The Environment tree: For more information, see "Mapping the environment tree" on page 1471

Note: Other message assembly components, such as the Exception list tree, are passed on unchanged. They cannot be modified in a message map.

Message body

You define the message body by defining a message model in any of the following ways:

- Define a message model by using a predefined message model.
- Define a message model by using a user-provided message model.
- Define a message model by adding user-defined elements

You can use message models for any messages that you want to include in a mapping. You can select the message model from your existing message models in your application, integration service, or library when you create a message map. The mapping facility supports message models that are provided in

DFDL schema and XML schema files, REST API swagger 2.0 documents, JSON schema draft 4, or MRM message sets. For more information, see "Input and output messages to a message map" on page 1276.

In a message map, input and output body objects are defined by reference to message models, which provide a definition of the message structure and type through the following components:

- Simple elements and attributes, which define the type, range, and default values
- Complex elements, which build the structure of the message
- Repeating simple or complex elements
- Other (embedded) messages

If your message model includes wildcards (`xsd:any`), you can use a **Cast** function to redefine these data elements to a global type or element from any message schema in your application. For more information, see "Casting elements in a message map" on page 1398. You can also qualify your wildcards by adding dynamically user-defined elements.

Creating a message map

When you create a new message map, you can select the model that defines the input and output messages for that map. You can select data type defined models from XML, DFDL or JSON schema, or Message Sets. You can also select predefined messages for SOAP, JSON, or BLOB. Alternatively, when you implement an operation in a REST API project and you create a message map from within the subflow for that operation, you can choose to create the map with inputs and outputs for that REST API operation defined automatically, including the path and query parameters. To create the message map with input and outputs defined automatically, select **Message map with the input and output for REST API operation *operation_name*** in the **Specify a new message map file** dialog. For more information, see "Creating a message map" on page 1382.

Editing a message map

You edit a message map in the Graphical Data Mapping editor. You can create, modify, or delete a message map.

The Graphical Data Mapping editor saves message maps as `.map` files.

For more information, see "Editing message maps" on page 1394.

Mapping behavior

By default, any message assembly component that is not included in the message map is copied from input to output unchanged.

To modify a message assembly component, you add the header or folder to the input and output message assembly in the message map, and provide transformations.

To delete a message assembly component, you add the component to the input message assembly, but not the output message assembly in the message map.

To create a message assembly component, you add the component only to the output message assembly in the message map.

Mapping operations

You can use any of the following transforms to map graphically your data in the Graphical Data Mapping editor:

- **Core mapping transforms**: You can use built-in structural and functional mapping operations to graphically construct the required message transformations to build the output message.
- **Custom transforms**: You can use custom transformations to define specialized transformations in XPath 2.0, Java, or ESQL functions.

- **XPath functions** (`fn:`*`functionName`*): You can use XPath 1.0 and XPath 2.0 functions to transform data in a message map.
- **Database transforms**:
  - You can use the **Select** transform to query one or more database tables, and retrieve data that you can use in the message map. You can use the data to set output element values, define conditions, or use as input to build other transforms conditions. Database tables can be set as extra outputs of a message map.
  - You can use a **database routine** transform to call a stored procedure from a database, and retrieve data that you can use in the message map. You can use the data to set output element values, define conditions, or use as input to build other transforms conditions.

    Note: For information about the support for stored procedures, see "Support for stored procedures" on page 1318.
  - You can use the **Insert** transform to add one new row of data, or multiple rows of data, into a database table.
  - You can use the **Update** transform to modify a row of data, or multiple rows of data, in a database table.
  - You can use the **Delete** transform to delete a row of data, or multiple rows of data, in a database table.
- **Cast** function: You can use the **Cast** function to cast schema types.

For more information, see "Transforms (Mapping operations)" on page 1280.

Local maps

You can use local maps as navigation aids. You view the map elements in a hierarchical way. Unlike submaps, local maps are not separate files and they are not reusable. They provide a way of breaking up a large map into nested groups of mapping elements and processing the complex elements of the whole message.

Deploying a message map

Message map files are deployed to the IBM App Connect Enterprise run time environment to enable them to be run in a message flow.

When you build and deploy a BAR file for an integration solution, the message map files are automatically included.

When you deploy independent resources, the BAR file editor provides a resource category to allow message maps to be selected for deployment.

If your message map file is used by multiple solutions, you might store the map in a shared library. Shared libraries that are referenced by applications must be deployed with or before the applications that refer to them. You can deploy the shared libraries directly to the integration server before you deploy the applications. Alternatively, you can include the applications and shared libraries in a BAR file and deploy the BAR file. If you update the message map in a shared library, those changes are available automatically to all applications that refer to that shared library.

Submaps
You can use a **submap** to use the same mapping transformation in multiple message maps.

You use submaps to define a set of mapping functions that you can reuse in multiple message maps.

A **submap** transform references another map. It calls or invokes a map from a separate file, which can be stored in a library, an application, an integration service, or an Integration project.

A submap can contain components of the message body only, such as global elements and global types. A submap does not contain Properties, message headers, or the Local Environment tree. The elements or types that define the input and outputs of a submap must be defined as globals in an XML or DFDL

message model. You cannot use user-defined elements as a submap input or output unless it is defined using a global type in an XML or DFDL schema. You cannot use a JSON data element defined from a JSON schema, or REST API swagger document, as a submap input or output.

For more information, see "Creating a submap" on page 1386.

When you use submaps, you must consider the following behavior:

- You can only use a submap to define transformations between DFDL or XML global elements or global types.
- A submap cannot be used to transform local anonymous complex types, that is, `xs:any` elements.
- A submap can be placed in any project that is visible to the main map that calls the submap.

Editing a submap

You edit a submap transform in the Graphical Data Mapping editor.

The Graphical Data Mapping editor saves submaps as `.map` files.

For more information, see "Graphical Data Mapping editor" on page 1266.

Input and output data to a submap

To configure the inputs and the output to a submap transform, you connect one or more inputs, and a single output.

The input and output message data for a submap must be defined by a user-provided message model in a schema file.

Note: You cannot use submap transforms with input or output from the following places:

- A user-defined element, unless its type is set to a global type in a DFDL or XML schema.
- A JSON data type defined in JSON schema or a REST API swagger document.

You must have message models for any messages that you want to include in a mapping. You can select the message model from your existing message models in your application, integration service, or library when you create a message map. The mapping facility supports message models that are provided in DFDL schema and XML schema files, REST API swagger 2.0 documents, JSON schema draft 4, or MRM message sets.

In a message map, input and output objects are defined by reference to message models. They provide a definition of the message structure and type through the following components:

- Simple elements and attributes, which define the type, range, and default values
- Complex elements, which build the structure of the message
- Repeating simple or complex elements
- Other (embedded) messages

If your message model includes wildcards (`xsd:any`), you can use a **Cast** function to redefine these data elements to a global type or element from any message schema in your application. For more information, see "Casting elements in a message map" on page 1398. You can also define dynamically a wildcard by adding user-defined elements.

Mapping operations

You can use any of the following transforms to map graphically your data in the Graphical Data Mapping editor:

- **Core mapping transforms**: You can use built-in structural and functional mapping operations to graphically construct the required message transformations to build the output message.
- **Custom transforms**: You can use custom transformations to define specialized transformations in XPath 2.0, Java, or ESQL functions.

- **XPath functions** (`fn:`*`functionName`*): You can use XPath 1.0 and XPath 2.0 functions to transform data in a message map.
- **Database transforms**:
  - You can use the **Select** transform to query one or more database tables, and retrieve data that you can use in the message map. You can use the data to set output element values, define conditions, or use it as input to build other transforms conditions. Database tables can be set as extra outputs of a message map.
  - You can use a **database routine** transform to call a stored procedure from a database, and retrieve data. You can use the data to set output element values, define conditions, or use as input to build other transforms conditions.

    Note: For information about the support for stored procedures, see "Support for stored procedures" on page 1318.
  - You can use the **Insert** transform to add one new row of data, or multiple rows of data, into a database table.
  - You can use the **Update** transform to modify a row of data, or multiple rows of data, in a database table.
  - You can use the **Delete** transform to delete a row of data, or multiple rows of data, in a database table.
- **Cast** function: You can use the **Cast** function to cast schema types.

For more information, see "Transforms (Mapping operations)" on page 1280.

Creating a submap

A submap can be referenced from other message maps.

When you construct your transformation map, you create a submap to group part of the message transformation. The submap must be in a project visible to the main map that they are called from.

To create a submap, you define a **Submap** transform between the input object and the output object in your message map. The submap can then be used to enable reuse of common transformations for sections of, or the whole of, the message.

You can also refactor existing transform logic into a submap from a local transform, by using the context menu action.

For more information, see "Calling a submap" on page 1540.

Reusing a submap

You can use a submap to reuse common data transformations.

You can reuse submaps in other solutions, and in other products that support graphical data maps.

Note: If you plan to reuse a submap across different products, read "Guidelines for developing reusable graphical data mapping assets" on page 1273.

Guidelines for developing reusable graphical data mapping assets
You can create graphical data maps (`.map` files) for reuse by other products that include the Graphical Data Mapping editor.

When you create graphical data maps that you want to reuse in other products, ensure that your graphical data maps conform to the following guidelines:

Build your reusable mapping structures as a submap

Many products use extra metadata to supplement the business message data, but a reusable transformation asset that is contained in a submap processes only the common business data.

Create a submap to contain the mapping structures that you want to reuse in other products, and then call this submap from a top-level mapping structure. You can use the Graphical Data Mapping editor to refactor a local map to a submap; see “Converting a local map into a submap” on page 1388.

For more information about submaps, see “Creating a submap” on page 1386.

Custom transforms can be only Custom Java and Custom XPath

Create custom transforms in your reusable submaps by using only **Custom Java** and **Custom XPath** transforms. Other custom transform types, such as Custom ESQL in IBM App Connect Enterprise, are product-specific, and cannot be used.

Considerations for mapping messages modeled in message sets
You can use the Graphical Data Mapping editor to transform XML messages that are defined in XML Schema (`.xsd` files). Message Set modeling in IBM App Connect Enterprise supports XML Schema extensions.

The Graphical Data Mapping editor supports both XML and text or binary messages that are modeled in IBM App Connect Enterprise message sets, with the following considerations:

- Message sets provide facilities for defining message composition. When these extensions are used to redefine a wildcard in a message, they are not shown in the message map. The Graphical Data Mapping editor provides equivalent facilities for modeling choice in schema wildcards by using the Cast function. For more information about the Cast function, see “Mapping xsd:any on an input or output message” on page 1397.

  When you create a message map or convert a message map that includes a schema wildcard from a message set with XML Schema extensions, you must manually add a Cast function from the wildcard to the required schema element.
- The message map requires the message set schema (`.xsdzip` file) to be deployed to run your message map. If your existing message set is used for text and binary formats only, you can deploy your message map with only a `.dictionary` file in the BAR file. In this case, you must modify the message set to additionally set the XMLNSC domain support option, so it is added to a BAR with both a `.dictionary` file and `.xsdzip` file. If this option is not set, a warning is displayed in the **Problems** view, along with a quick fix action.

Designing a message map

You can use a message map to graphically transform, route, or update an external system. For best performance and capability, you must design it to include the most appropriate transforms.

Procedure

Consider the following guidance to design a message map:

1. Design the data model of your input and output message per the solution requirements.

   The function of a message map is completely dependent on the data models that define the input and the output message structures of a map. At run time, the Graphical Data Mapping engine accesses the input message with the assumption that it conforms exactly to the provided input schema model. Additionally, it must account for all possible states of the data when executing the transformations that you have defined in the map against the data models. You might or might not have control over the data models in your solution. If you can influence the data model, these are some of the key points to consider:

   a) If you are working with XML schema models, you must always enable schema validation when you are developing a message map. The schema validation ensures that the input message to the map matches the model that has been used to define the map, and prevents invalid mapping due to any mismatch.

      Validation is the process of checking the structure of a message, and optionally the values within, based on a description called a schema. The Mapping node relies on schema definitions, but it does not enforce them. If the input message does not conform to the schema being used, the output

you expect to see might not be produced. During the development of your integration solutions, it is recommended that you enable validation. However, for other environments such as test or production, you should leave validation on only if your solution requires it, because of the extra processing involved. For more information, see Validating messages.

If you are working with DFDL schema models, and the message has been received by an input node, the input to the map is validated by the DFDL parser. If the DFDL tree is constructed by other means, ensure that it is valid by running test serialize, which you can do by using a trace node.

If you are working with JSON models, there is currently no internal support for JSON schema validation. Ensure that your message is valid by using an external tool.

b) Set the cardinality of each element in a data model to specific values whenever possible.

When you define a logical model, you can configure the cardinality of each element by setting the **minoccurs** and the **maxoccurs** properties.

Avoid, whenever possible, configuring maximum flexibility unless actually required. Only set **minoccurs** to 0, when an element needs to be optional. Only set **maxoccurs** above 1 if the element will actually repeat.

c) Define an element as nillable when you know that the application will need to handle out of bound value in the data.

2. Identify the type of message map.

a) Use a message map to graphically transform, route, and enrich a message. You can use a message map to modify data in a database system.

For more information, see “Message maps” on page 1268.

b) Use a submap to define a set of mapping functions that you can reuse in multiple message maps.

For more information, see “Submaps” on page 1271.

For example, the Mapping node invokes a map that deals with a message assembly. You can put the mapping for common parts of the message data into a submap to enable reuse.

3. Identify the input and output components to a message map.

- You can select an XML schema, DFDL schema, or message set to define the message body.
- You can choose any of the following message assembly components: the Properties tree, the local environment tree, or the environment tree.
- You can add database tables.

You should only include a component when you need to read data or write data:

- To read elements of an input component, add the component to the input message assembly only. The Graphical Data Mapping editor passes to the output the input component unchanged.
- To modify elements of an input component, add the component to the input message assembly and to the output message assembly. Then, define transforms between its elements.
- To initialize an input component, that is, to create a new component in your output message, add the component only to the output message assembly.
- To add an input component, add the component to the output message assembly and populate at least one field. The Graphical Data Mapping editor creates a new output structure containing the results of your transformations.
- To delete an input component from the input message, add the component to the output message assembly and do not set any field.

For more information, see “Choosing a mapping action” on page 1450.

4. For each output element, identify the transform and the input elements required to calculate its value.

   When the transformation of an element from input to output becomes more than just a simple **Move**, or type conversion (**xs:type**), you can call on the full set of standard XPath 2.0 operators and functions to manipulate the data as required.

   The Graphical Data Map editor offers the XPath functions as transform types in the pick list as well as in the content assist (Ctrl-space) when you edit expressions and conditions.

   - “Choosing a transform to set the value of a simple type output element” on page 1358
   - “Choosing a transform to set the value of a complex output element” on page 1360
   - “Choosing a transform to map repeating elements” on page 1362
   - “Choosing a transform to concatenate input data” on page 1364
   - “Choosing a transform to perform an arithmetic operation” on page 1366
   - “Choosing a transform to define a conditional mapping” on page 1367
   - “Choosing a transform to map an input message to multiple output messages” on page 1368
5. Define a conditional expression for each transform to determine at run time whether the transform is applied or not.

   For more information, see “Defining an XPath conditional expression for a transform” on page 1433.
6. Use structural transforms to enhance the readability and maintenance of your map and, in some situations, reduce the runtime overheads of your map. Use the transforms **If**, **Local Map** or **Submap**, and **For each** to group transforms.

   Using structural transforms has the following advantages:

   a. You can define a conditional expression that determines whether a nested map is applied at run time. For example, if a set of child fields should only be mapped dependent on some attribute of the folder, place all the child elements inside an **If** transform. The nested mappings are executed only when the condition evaluates to **true**.

   b. You can use a **Local Map** transform between the parent folders that are defined as optional (<code>minOccurs=0</code>) instead of mapping directly between child elements in the folders. By using a **Local Map**, you create a more efficient transformation because, by default, the nested mapping is executed only if the primary input is optional (`minOccurs=0`) and is available at run time. Alternatively, you can add your own condition, which can also use additional supplemental inputs.

   c. When you use a **Local Map** transform, you can convert the map to a submap if the need for reuse comes at a later stage. You can use an action in the Graphical Data Mapping editor to re-factor the **Local Map** into a **Submap**.

   d. You can use the Auto map wizard (automap) to create **Move** transforms to create mappings from input to output elements within structural transforms that are based on some correlation of the names of the input and output elements. By using the structural transform, the scope of the automap is reduced, which enables a better match to be achieved.

   For more information, see “Using nested maps” on page 1371.
7. When you need to process data, rather than just move it from source to target, use custom transforms to define specialized transformations.

   a) “Custom XPath” on page 1315

   b) “Custom Java” on page 1312

   c) “Custom ESQL” on page 1308

   From a performance point of view, it is recommended that you use **XPath** transforms or the **Custom XPath** transform as your first choice, then **Custom Java**. You can also use **Custom ESQL**.

Input and output messages to a message map

In a message map, you must define a model for the input message and an output message. You can choose from a predefined message format, or you can use an XML, DFDL, or JSON schema message

model, to define the structure of the data and provide information about the data type. Alternatively, you can define it dynamically in the map by using the **Add User-Defined** function.

When a message arrives to IBM App Connect Enterprise, a parser is called. Each parser is suited for a particular type of message, and is known as a message domain. A parser converts the bit stream of an input message to an internal format. A parser is also called when a logical tree that represents an output message is converted into a bit stream.

Note: The data structure that you define in a message map for an input or an output message is the IBM App Connect Enterprise internal representation of the message.

Message domains

The following message domains are supported in a message map:

DFDL
XMLNSC
SOAP
DataObject
BLOB
JSON
MRM

Message assembly

In a message map, the *source message assembly* describes the input message and the *target message assembly* describes the output message.

A message assembly includes the properties tree, any relevant headers, the local environment tree, the Environment tree, and the message body.

- When you create a top-level message map, only the Properties folder is initially included. A **Move** transformation from the input Properties folder to the output Properties folder is created by default where all input values are copied to the corresponding output values unchanged.
- The structure of the Properties folder, the transport headers, and the local environment tree are predefined in IBM App Connect Enterprise.
- You can define the local environment tree **Variables** folder structure by using the **Cast** function. You can also define elements of the **Variables** folder by using the **Add User-Defined** function.
- You can define one or more elements under the **Variables** folder of the Environment tree. You can use the function to model Environment elements. Elements that are not defined in the map are maintained unchanged.
- The input message body is defined by associating an input message model such as a DFDL schema, a JSON schema, or an XML schema. You can also define the input message body by using the **Add User Defined** function, for example, to define the structure of a JSON message.
- The output message body is defined by associating an output message model such as a DFDL schema, a JSON schema, or an XML schema. You can also define the output message body by using the **Add User Defined** function, for example, to define the message body structure of a JSON message.

The **Output domain** property target message assembly is set to define the message domain in which an output message is to be built, based on the message type that you selected for the map output. You can change it if necessary.

The message map uses the schema types of the output elements to create and set the elements of the output message tree.

Message models

The following message models are supported in a message map:

- Predefined message model
- XML schema-based message model
- DFDL schema-based message model for text and binary formats
- JSON schema-based model, from a REST API swagger document, or a JSON schema in a shared library
- Schemaless message model, that is, a message model of a message that has a well-defined format but no schema definition available as an `xsd` file.

You can select any of the following supplied message models as your input or output message format:

- **SOAP_Domain_Msg {}**: Use this message model to handle messages in the SOAP domain; it allows you to add mapping casts to define the content of the SOAP body and header by using a schema model.
- **JSON Object Message**: Use this message model for JSON object data in the JSON domain, by using the **Add User Defined** function.
- **JSON Array Message**: Use this message model for JSON array data in the JSON domain, by using the **Add User Defined** function.
- **BLOB {}**: Use this message model to handle messages in the BLOB domain, or when you do not need to process any message body data.

Advanced XML schema structures valid in input and output messages
You can use several advanced schema structures in graphical data maps.

You can use any of the following XML features in message models that are defined as inputs or outputs to the Graphical Data Mapping editor:

- "Substitution groups" on page 1278
- "Wildcards (xsd:any)" on page 1278
- "Derived types" on page 1279
- "List types (xs:list)" on page 1279
- "Union types (xs:union)" on page 1279

Substitution groups

A *substitution group* is an XML schema feature that provides a means of substituting one element for another in an XML message.

- The element that can be substituted is called the **head** element.
- The **substitution group** is the list of elements that can be used in its place.

The head element and any mapped substitutions are shown by default in the Graphical Data Mapping editor. The mapped substitutions are listed beneath the head element.

You create mappings to or from members of substitution groups in the same way as you map other elements.

Wildcards (`xsd:any`)

You can use *wildcards* to create open content models. You can define `xsd:any` and `xsd:anyAttribute` wildcards.

Wildcards are characterized for the following attributes:

- The namespace attribute: You can use this attribute to specify the namespace that the elements or attributes that match the wildcard can come from.
- The processContents attribute: You can use this attribute to specify how the XML content matched by the wildcard is validated.

When you use wildcards in the Graphical Data Mapping editor, you must consider the following behavior:

- You can wire a `xs:any` or a `xs:anyAttribute` as the input or output of a **Submap** transform. Then, when you configure the **Submap** transform, you can define that input or output to be a particular type.

- A wildcard element can be instantiated only with another element.
- A wildcard attribute can be instantiated only with another attribute.
- You can use a global element or attribute as a wildcard replacement.

For more information, see "Mapping xsd:any on an input or output message" on page 1397.

Derived types

A *derived type* is a data type that is related to another data type known as the base type or super type.

In a message map, you can cast a base type to a derived type or extension type. You can define transformations between subtypes of a data type.

When you use derived types in the Graphical Data Mapping editor, you must consider the following behavior:

- For an element of a specific type, the base type and the mapped derived types are shown by default. All attributes and elements of the base and derived types are displayed.
- You create mappings to or from a derived type and its elements in the same way that you map any base type and its elements.
- When you map an input element to an output element with a derived schema type or an extension schema type, the created output element is set with the relevant `xsi:type` attribute for that schema type.

List types (`xs:list`)

You can use `xs:list` to define a simple type element as a list of values of a specified data type. For example:

```
<xs:simpleType name='CustomerName'>
 <xs:list itemType='string'/>
</xs:simpleType>
```

When you use `xs:list` in the Graphical Data Mapping editor, you must consider the following behavior:

- You map a simple type element that is defined as a list of values in the same way that you would map any other simple type attribute or element.

Union types (`xs:union`)

You can use `xs:union` to define a simple type as a collection of values from specified simple data types. It allows a value to conform to any one of several different simple types. For example:

```
<xs:element name="zipUnion">
 <xs:simpleType>
  <xs:union memberTypes="USStateName StateID"/>
 </xs:simpleType>
</xs:element>
<xs:element name="USStateName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="StateID">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

When you use `xs:union` in the Graphical Data Mapping editor, you must consider the following behavior:

- You map a simple type element that is defined as a collection of values in the same way that you would map any other simple type attribute or element.
- You must check that the value of an element is processed with the correct type when you use an element that is defined with a union type.

  For example, you can use the expression **$zipUnion != ''** to check that the value of the element **zipUnion** is not an empty string. The Graphical Data Mapping editor might fail at run time with **BIP3949E** if the element **zipUnion** is not of string type.

  ```
  BIP3949E: An error occurred during the processing of a message.
  ```

  To prevent this error, you can explicitly define the type of the element by using the **xs:string()** function:

  ```
  xs:string($zipUnion) != ''
  ```

  For more information, see "Cast type (xs:type)" on page 1299.

Transforms (Mapping operations)
In the Graphical Data Mapping editor, you can define mapping operations such as transforms, cast functions, or XPath 2.0 functions. Mapping operations define the transformation actions on input data, and set the result to the output element.

Mapping operations to transform graphically your data

You can use any of the following transforms to map graphically your data in the Graphical Data Mapping editor:

- **Core mapping transforms**: You can use built-in structural and functional mapping operations to graphically construct the required message transformations to build the output message. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.
- **Custom transforms**: You can define specialized transformations to build custom ESQL functions, XPath 2.0, or Java. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.
- **XPath functions**: You can use XPath 1.0 and XPath 2.0 functions to transform data in a message map. You can also build compound XPath expressions by using the **Custom XPath** transform; see "Custom XPath" on page 1315.

  All XPath 2.0 functions are supported in the form `fn:`*`functionName`*.

  For more information about XPath, see the online document W3C XML Path Language (XPath) 2.0.
- **App Connect Enterprise mapping functions**. You can use a set of IBM App Connect Enterprise mapping functions (`iib:`*`functionName`*), for example, to access a user-defined property value. For more information, see "Custom XPath" on page 1315.
- **Database transforms**:
  - You can use the **Select** transform to query one or more database tables, and retrieve data. You can use the data in the message map to set output element values, define conditions, or use as input to build other transforms conditions. Database tables can be set as extra outputs of a message map. For more information, see "Selecting data from a table" on page 1523.
  - You can use a **database routine** transform to call a stored procedure from a database, and retrieve data. You can use the data in the message map to set output element values, define conditions, or use as input to build other transforms conditions.

    Note: For information on the support for stored procedures, see "Support for stored procedures" on page 1318.

- **Cache transforms**: By using **Cache** transforms you can interact with data that is stored in a global cache. For more information, see "Accessing a global cache by using a Mapping node" on page 1514.

  You can use the following transforms:

  - **Cache Put** transform: You use the **Cache Put** transform to add a key-value pair into the global cache. For more information, see "Adding key-value pairs to the global cache by using a Mapping node" on page 1515.
  - **Cache Get** transform: You use the **Cache Get** transform to retrieve a value from the global cache by providing the associated key. For more information, see "Retrieving a value from the global cache by using a Mapping node" on page 1517.
  - **Cache Remove** transform: You use the **Cache Remove** transform to remove a key-value pair from the global cache. For more information, see "Removing a key-value pair from the global cache by using a Mapping node" on page 1519.

Mapping operations to modify data in a database

Database tables can be set as more outputs of a message map.

You can use any of the following transforms to modify data in a database:

- **Insert** transform: You use the **Insert** transform to add one new row of data, or multiple rows of data, into a database table. For more information, see "Inserting data into a table" on page 1525.
- **Update** transform: You use the **Update** transform to modify a row of data, or multiple rows of data, in a database table. For more information, see "Updating data in a table" on page 1527.
- **Delete** transform: You use the **Delete** transform to delete a row of data, or multiple rows of data, in a database table. For more information, see "Deleting data from a table" on page 1528.
- **Database routine** transform: You use a **database routine** transform to call a stored procedure or user-defined function from a database to insert, delete, or update data in one database table. For more information, see "Calling a stored procedure from a map" on page 1532.

At design time, you must have a database definition file (`.dbm` file) in an available Data Design project for each database that you want to access. A data definition file contains one connection per database system.

At run time, you must have a JDBC connection of Type 4 defined for each database that your message map uses. You must configure a JDBC Providers policy per database. The JDBC Providers policy name for a runtime database must be the same name as the development database name that you use in your message map.

For more information, see "Modifying data in a database by using mapping " on page 1525.

Cast function to define a schema type

In the Graphical Data Mapping editor, you can use the **Cast** function to cast schema types.

You must cast a schema in any of the following instances:

- Cast the value that you assign to an output element so it matches the output element schema definition type.
- Cast the value of an element that you use as a parameter to a function where the parameter is of a different type.
- Cast the value of an element that you use as a condition on a transform where the type is different.
- Cast the value of an element when you work with a database, and the types differ.

To cast a schema, you can use the `xs:`*`castOperation`* functions, where *castOperation* is the name of the cast function.

Mapping behavior driven by the type of operation

When the transformations in the message map are constructed, the values for output message elements can be derived from any of the following components:

- Input message elements, through any of the following mapping operations:
  - **Move**, **Convert**, and other built-in transforms in the Graphical Data Mapping editor.
  - XPath 2.0 functions (prefix `fn:`). All XPath 2.0 functions are supported by the Mapping node. For more information about XPath, see W3C XML Path Language (XPath) 2.0.
  - Database input by using a database **Select** transform.
  - Schema type casts. For more information, see "Mapping xsd:any on an input or output message" on page 1397.
- Extra functions, which allow multiple input values to produce the output value, such as **concat** and **join**.
- The result of database **Select**, **Insert**, **Update**, **Delete**, and **Database routine** transforms.
- Constant values, through an **Assign** operation that uses a supplied value.
- Custom functions, user-defined XPath, Java, or ESQL.

The logic to derive values can be simple or complex. In addition to the transformation operations that set an output value, structural transforms are provided to enable conditional statements, loops, and nesting of transform logic into local maps.

For information about the supported transform types, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Transform types in the Graphical Data Mapping editor
In the Graphical Data Mapping editor, you can map elements and attributes between the input and output objects. You can apply a transform to the mapping that specifies the action to be performed on the input data. The result of the transform is stored in the output element.

The following table shows the standard mapping transforms that are provided by the Graphical Data Mapping editor:

Table 44. Core mapping transforms in the Graphical Data Mapping editor:

Transform	Description
"Assign" on page 1289	Sets a value in the output element. There is no input element. Column values set via **Assign** transform will always be passed as character string.
"Setting the value of a simple output element to a default or fixed value" on page 1485	Sets a specific value type in the output element. **Cast** can also move and convert an input element to become a specific value type in the output element.
"Concat" on page 1301	Creates a string concatenation that allows you to retrieve data from two or more entities and link them into a single result.
"Convert" on page 1303	Copies the input element to the output element and changes the type. The transform takes a single simple input and creates a single simple output with a different type.
"Create" on page 1304	Creates an empty element, a nil element, or a simple type element by using a default value that is based on the element's type.
"Custom XPath" on page 1315	Enables you to enter any XPath expressions and built-in IBM Integration Bus functions to be used in the transform.

Table 44. Core mapping transforms in the Graphical Data Mapping editor: (continued)	
Transform	**Description**
"Custom Java" on page 1312	Enables you to enter your own Java code to be used in the transform.
"Custom ESQL" on page 1308	Enables you to enter your own ESQL code to be used in the transform.
"Move" on page 1335	Copies data from the input element to the output element.
"Normalize" on page 1336	Normalizes the input string by removing white space such as spaces, tabs, and returns, and moves the resulting normalized string to the output element.
"Substring" on page 1340	Extracts information as required, and moves the extracted string to the output element.
"Task" on page 1341	Describes a manual task or point of concern that might need to be reviewed or resolved before a message map can be used in your solution.
"Built-in XPath transforms" on page 1342	All XPath 2.0 functions are supported, in the form `fn:<function_name>`.

In addition to the core mapping transforms, several structural transforms are provided. The structural transforms control how nested elements are displayed in the Graphical Data Mapping editor, but they have no effect on the data itself. The structural transforms are described in the following table:

Table 45. Structural mapping transforms in the Graphical Data Mapping editor:	
Transform	**Description**
"Append" on page 1285	Appends occurrences of an output array in the order of the inputs.
"Remove" on page 1336	Removes an element.
"For Each" on page 1320	Iterates over an input array element (either a simple type or a complex type).
"Group" on page 1328	Takes a single input array and produces a set of nested output arrays that collate elements of the input array.
"If, Else if, and Else" on page 1331	You can control the flow of the mapping by setting conditions.
"Join" on page 1332	Joins elements from two or more inputs.
"Local map" on page 1334	Provides a hierarchical view of element transforms in the message map.
"Submap" on page 1339	References another map. It calls a map from this map file or another map file, which can be stored in a library, an application, an integration service, or an Integration project.

The following table shows the database transforms that are provided by the Graphical Data Mapping editor:

Table 46. Database transforms in the Graphical Data Mapping editor:	
Transform	**Description**
"Database Routine" on page 1318	Calls a stored procedure or user-defined function from a database.
"Delete" on page 1319	Deletes one or more rows in a database table that is matched by a Where clause.
"Failure" on page 1319	Enables the map to take-on error handling for any exceptions that are raised by the database server in a database transform, instead of having such exceptions stop the map and be reported.
"Insert" on page 1332	Inserts a row into a database table.
"Return" on page 1338	Enables extra processing after a successful Insert, Update, or Delete database operation, or Database Routine call. Provides the results from the database operation or call as inputs.
"Select" on page 1338	Retrieves data from rows in a database table, so that the data can be used as input in a message map.
"Update" on page 1342	Updates one or more rows in a database table that is matched by a Where clause with a single set of data values.

You can use **Cache** transforms to interact with data that is stored in a global cache. The **Cache** transforms are described in the following table:

*Table 47. **Cache** transforms in the Graphical Data Mapping editor:*	
Transform	**Description**
"Cache Put" on page 1293	Adds a key-value pair to a map in a global cache.
"Cache Get" on page 1291	Gets a value from a map in a global cache, by providing the key that is associated with the value.
"Cache Remove" on page 1296	Removes a key-value pair from a map in a global cache.
"Cache Return" on page 1297	Returns or transforms values if the **Cache** transform succeeds.
"Cache Failure" on page 1290	Returns or transforms values if the **Cache** transform fails.

You can use built-in IBM App Connect Enterprise functions to retrieve IBM App Connect Enterprise data and perform data-type conversion and formatting. The `iib:` functions are described in the following table, and are available through content assist in the "Custom XPath" on page 1315 transform:

Table 48. Built-in App Connect Enterprise functions in the Graphical Data Mapping editor:	
Function	**Description**
iib:getUserDefinedProperty('*propertyName*')	Access user-defined properties from a Mapping node
iib:hexBinaryValue($<var>)	Convert the value to a hex binary format string

Table 48. Built-in App Connect Enterprise functions in the Graphical Data Mapping editor: (continued)	
Function	**Description**
iib:base64BinaryValue($<var>)	Convert the value to a base64 binary format string
iib:nullValue()	Create a null/nil output element
iib:uuidValue()	Create a new UUID value

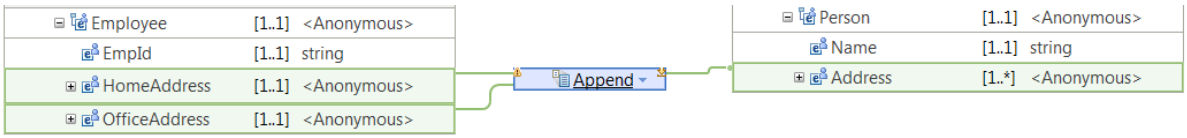
Append
You can use the **Append** transform to create an output array in the order of the inputs.

Overview

The **Append** transform is not available in the list of available transforms until you wire at least two inputs to a transform.

To build an output array of N elements where you have N inputs available in a flat structure, you wire each input to the **Append** transform.

For example, the following figure shows two flat structures, *HomeAddress* and *OfficeAddress* wired into an **Append** transform to create the *Address* array.



To build an output array of N elements where you have less than N unique inputs to wire into the **Append** transform, you can wire the same input multiple times.

If your input to the **Append** transform is repeating, you can use the **For Each** transform.

The only way to control the order of populating an output array is by using the **Append** transform. You must provide input connections to the **Append** transform and set their order in the **Properties** page.

Note: You must use the **For Each** transform, instead of the **Append** transform, when you need to create an output array where the input element is repeating. In the nested map of a **For Each** transform, you can use an **If** transform to define the condition under which the **Append** transform is applied for each instance of the input element.

Inputs to the Append transform

The **Append** transform takes multiple inputs of either simple type or complex type.

Inputs of single, non-repeating elements are also allowed. Each single input element adds an extra occurrence to the output array.

You wire inputs to the **Append** transform as **primary** connections.

You can wire the same input into an **Append** transform more than once if you need to produce more than one array instance from a single input.

The **Append** transform iterates over multiple inputs in the specified order to append data. The order of inputs into the transform is recognized, and is set in the **Order** property page.

The **Append** transform provides a nested transform for each input in the output array.

The nested transforms are performed for each input sequentially, producing occurrences of the output array. First over all elements in the first input, then over all elements in the second input.

When you connect a repeating input, each instance adds an extra occurrence to the output array. You can use the **Cardinality** property page on the transform to specify a subset of indexes the transform should process. The first index element is 1.

Output of the Append transform

The output of an **Append** transform can be a simple type array or a complex type array.

The output array size is the sum of the input elements wired to the **Append** transform.

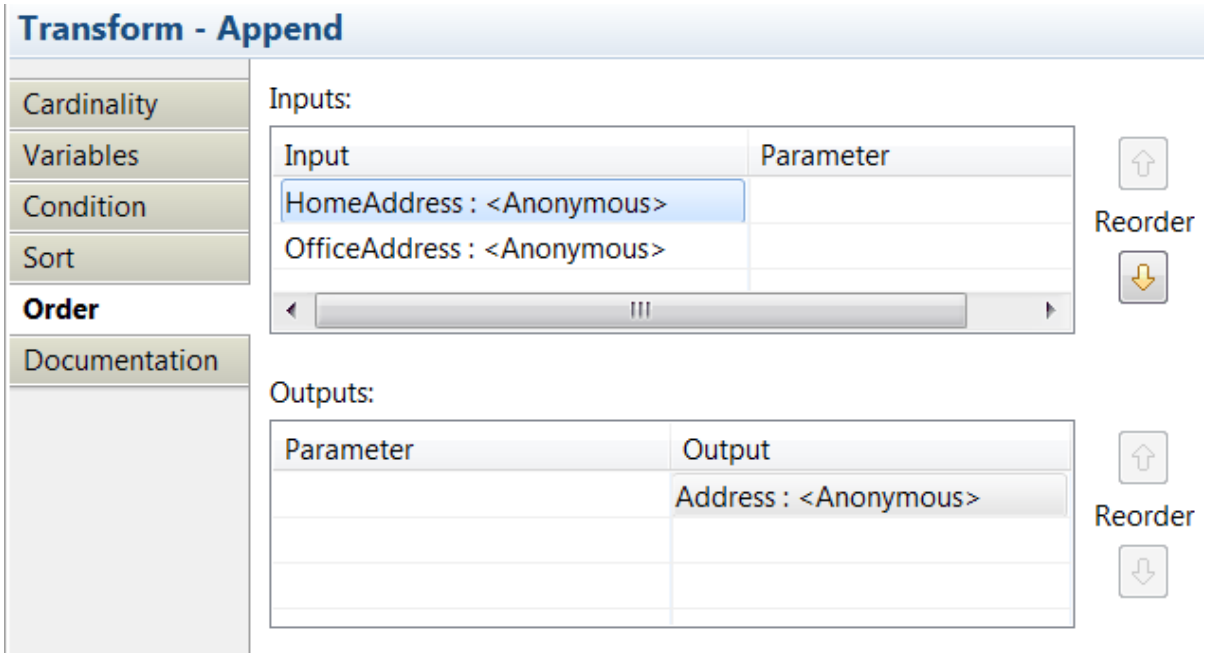
Note: You can define any number of inputs to the **Append** transform. The Graphical Data Mapping editor does not validate the number of inputs. You must ensure that the number of wired inputs to the **Append** transform correspond to the value configured in the **Maximum occurrence** property of the output array.

Order of the inputs

By default, the order of the inputs to the **Append** transform is the order in which you wire the inputs.

You can modify the order by reordering the inputs in the **Order** tab of the transform properties.

Note: You can only order inputs that are connected with a primary connection to the transform.



Inside the nested map

When you define a transform inside the nested map of an **Append** transform, you can only connect one input element to any transform that you define.

The Graphical Data Mapping editor reports error **CWMSL259E** if you connect multiple inputs to a transform inside the nested map.

If your input element is a repeating element, you can only use a **For Each** transform or an **If** transform on the repeating input.

- You can use the **For Each** transform to build an output instance for every index that is applied. The **Cardinality** and **Filter Inputs** properties determine which indexes of the repeating input are applied.
- You can use the **If** transform to build a single output instance. The **If** condition determines which index of the repeating element is applied to calculate the output value.

If the input element and the output element to an **Append** transform have a global type, you can use a **Submap** instead of a **Local map**.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Append** transform. You can then use one or more of these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Note:

- You can use only inputs that are connected with a primary connection to the transform.
- All inputs will be present in the nested map of the **Append** transform and by default mapped to produce an occurrence of the output.

If you need an input element to define a condition and you do not want the input element to be part of the output array, complete the following steps:

1. Connect the input element with a primary connection to the **Append** transform.
2. Define your condition.
3. In the nested map associated with the **Append** transform, a **Local Map** is added. Within the **Local Map**, a **Move** transform is defined for each input element. Delete the **Move** transform associated with the input element so that the input element is not appended to the output value.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

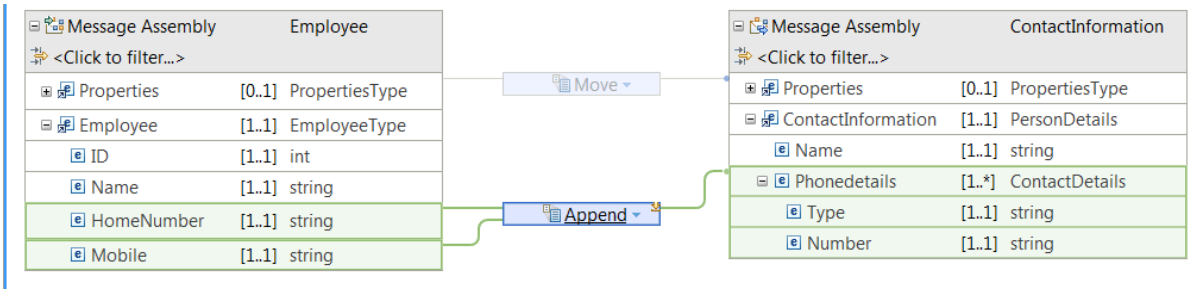
If your conditional expression requires the value of an input element that is not used in the **Append** transform to set the output element, you must use the **If** transform to define when the transform is applied at run time. For more information, see “If, Else if, and Else” on page 1331.

Example

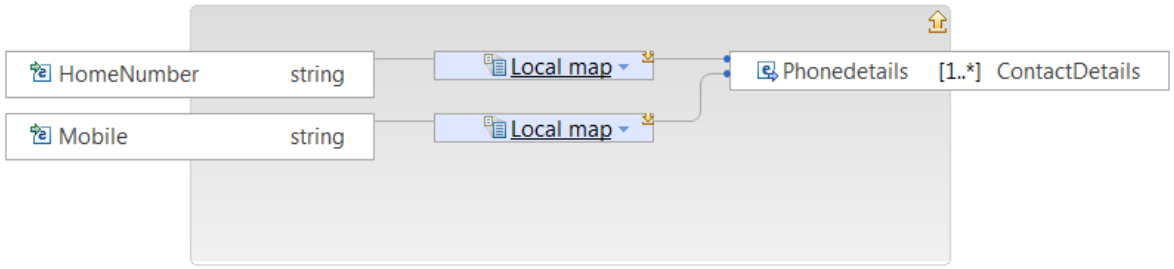
This example shows how to create an output array with two indexes by using the **Append** transform.

You connect two input elements (**HomeNumber** and **Mobile**) to the **Append** transform. These inputs will be used to set the value of multiple indexes in the output repeating element **Phonedetails**.

You also define a connection from the **Append** transform to the repeating element **Phonedetails**.

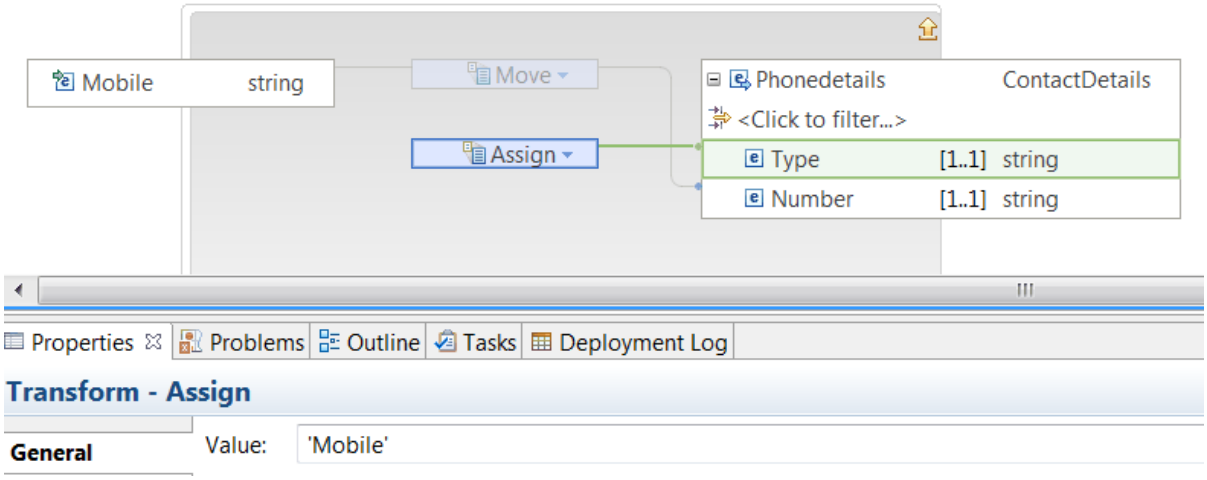


When you open the nested map associated to the **Append** transform, you get two **Local Map** transforms, one per input element.

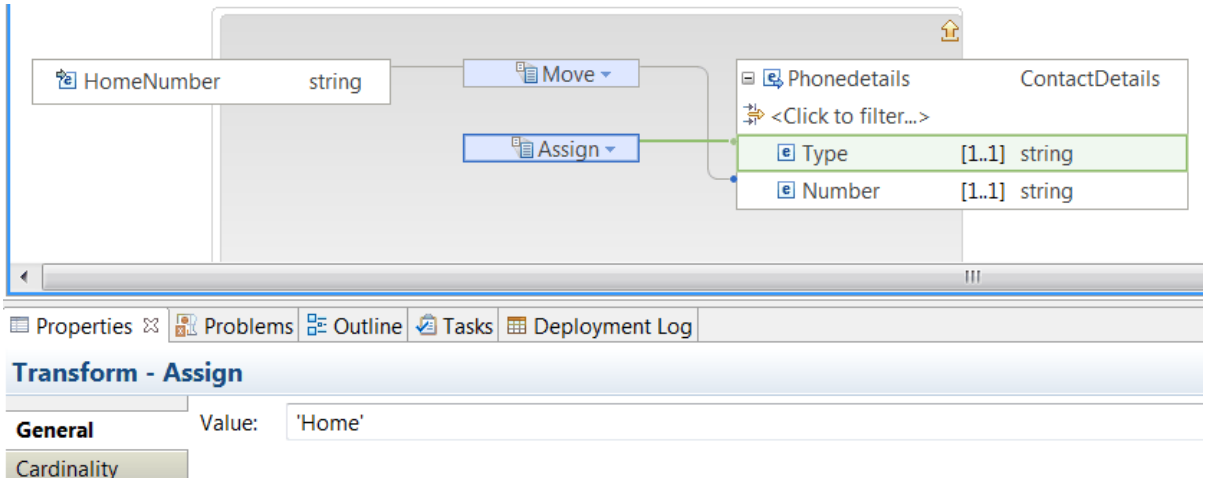


Inside each **Local Map** transform, you define the transformation logic that sets the output values for one index of the output repeating structure **Phonedetails**.

The following figure shows the nested map associated with the **Local Map** transform for the input element **Mobile**:

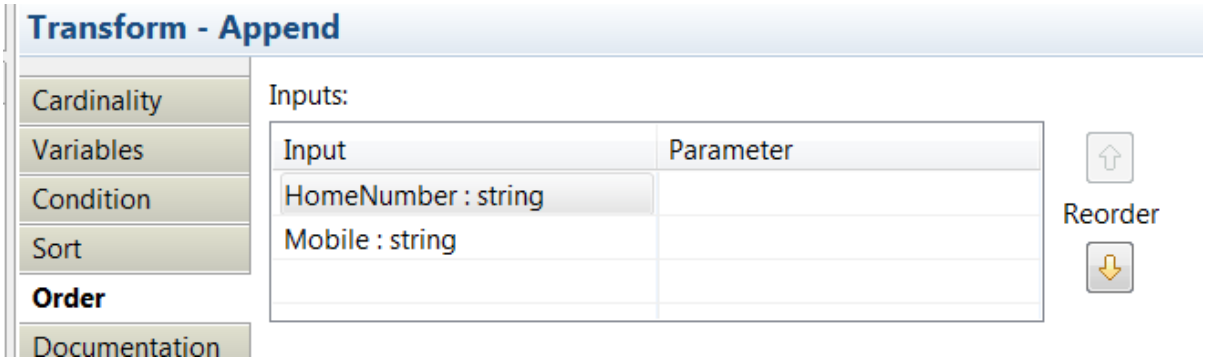


The following figure shows the nested map associated with the **Local Map** transform for the input element **HomeNumber**:



You can configure the order in which the indexes are created. You set the order in which indexes are created in the **Order** tab of the Properties view of the **Append** transform.

In this example, the first index contains the information of the home number. The second index contains the information of the mobile number.



Assign
You can use the **Assign** transform to set the value of an output element to a constant or fixed value.

Overview

The **Assign** transform sets a value that can be a fixed value or it can be the result of a function that has no input; for example, a current date function.

You cannot use the value of an input element to set the value of an output element with the **Assign** transform.

The output element is set to a constant value.

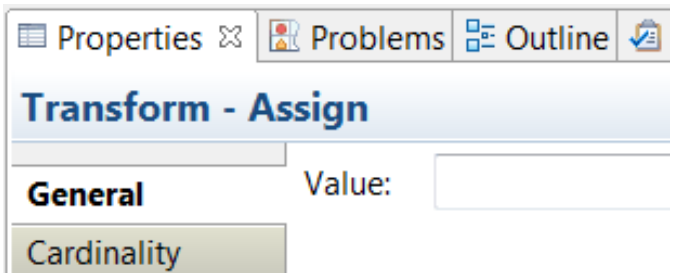
The output element must be a simple type element.

For more information about assigning a value to an output element, see "Setting the value of an output element to a simple data type" on page 1481.

Assign a value

You must assign a value in the **Value** field that is located in the **General** tab of the **Assign** transform **Properties** view.

If you do not specify a value, an empty element is created.



Define when the transform is applied at run time

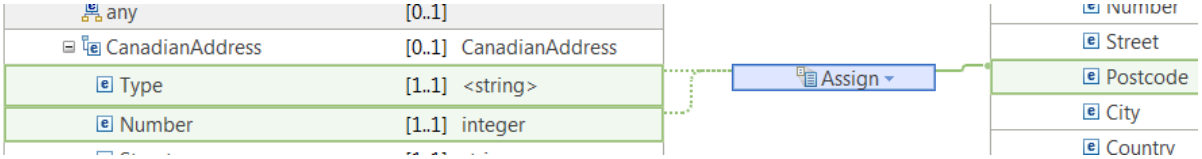
You can define multiple connections between input elements and the **Assign** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

To add inputs to the transform, you can define **supplement** connections between input elements and the **Assign** transform.



When you define more connections, you can configure the following properties to define how those inputs are displayed in the message map:

- **Sort**: You can sort the inputs to the transform by ascending order, descending order, case order, or data order.
- **Order**: You can display the order of input connections to a transform. You can reorder them.

Note: You only can sort and order inputs that are connected with a primary connection to the transform.

Default values

The following table lists the default values set when you use the **Assign** transform:

*Table 49. Default values set by using the **Assign** transform*

Type	Default value
string	Empty string
dateTime	`2002-01-01T11:00:00`
Boolean	`false`
decimal	`0.0`
double	`0.0`
hexBinary	`00`
long	`0`
duration	`P1Y`
time	`00:00:00`
date	`2002-01-01`

Cache Failure

You can use the **Cache Failure** transform to handle exceptions that might occur if a **Cache** transform fails.

Overview

Include a **Cache Failure** transform in your **Cache Put**, **Cache Get**, or **Cache Remove** transform group to specify a nested mapping that is called if the **Cache** transform fails. If you include a **Cache Failure** transform, exception errors are passed to the nested mapping instead of stopping the message map.

Note: If the conditions that are required to call a **Cache** transform are not met, the map stops and generates a mapping exception that cannot be handled by a **Cache Failure** transform.

A **Cache Failure** transform is an optional transform that can be added or removed as required

A **Cache Failure** transform does not perform any transformation. The **Cache Failure** nested mapping provides predefined input elements that provide data from an exception that is encountered when the cache operation runs. You can also map elements from the input tree to the **Cache Failure** transform, and connect the **Cache Failure** transform to one or more output elements. You can then use the nested map to create or transform values and map them to the output elements that you connected to the **Cache Failure** transform.

Note: If you add a **Cache Failure** transform but do not include any mapping in the nested map, the details of any exceptions in the associated cache operation are not reported anywhere; instead, the map displays a warning.

Predefined elements

The **Cache Failure** nested mapping has the following predefined input elements:

Element	Description
ID	The identifying code of the exception. This is the "BIP" code of the `MbException`.
Message	The message that is associated with the exception. For example: `A duplicate key was encountered while interacting with map 'SYSTEM.BROKER.DEFAULTMAP'. A client attempted operation 'put' with map 'SYSTEM.BROKER.DEFAULTMAP' from grid 'WMB', using key 'MyKey'. This attempt failed because an entry already exists with the same key. This operation requires that a key of the same name does not already exist in the map. : Duplicate key exception`
Inserts	The details that are specific to the exception. For example, in the previous message, the following entities are the inserts: • The cache map name: `SYSTEM.BROKER.DEFAULTMAP` • The operation: `put` • The cache name: `WMB` • The key: `MyKey`

Run time behavior

The handling of **Cache** transform exceptions is determined by the configuration of the corresponding **Cache Failure** transform:

- If the **Cache Failure** transform is present in the **Cache** transform group, and is connected to one or more output objects, the exception is caught and processed by the **Cache Failure** transform.
- If the **Cache Failure** transform is present in the **Cache** transform group but is not connected to any output objects, the exception is caught by the **Cache Failure** transform, and is ignored.
- If the **Cache Failure** transform is not present in the **Cache** transform group and the **Cache** transform generates an exception, the exception is handled by the Mapping node in your message flow in the same way as other message flow exceptions.

Cache Get

You can use the **Cache Get** transform to get a value from the global cache, by providing the key that is associated with the value.

Overview

The **Cache Get** transform accepts only input connections, which provide values for the parameters in the nested mapping.

You can assign or map only single values of type simple value to the key. If a mapped element is not of the appropriate data type, you can use the Cast type (**xs:type**) transform to assign the data type. If the element can contain multiple values, set the Cardinality property to ensure that a single value is mapped to the key; see "Configuring the cardinality of a user-defined element" on page 1412.

When you add a **Cache Get** transform to your graphical map, a **Cache Return** transform is automatically included; see "Cache Return" on page 1297. The **Cache Return** transform provides a single predefined input element, which contains the value from the key-value pair that is associated with the key that is mapped to the **Cache Get** transform. You can connect the **Cache Return** transform to any output element to map the data that is returned from the cache. You can also map elements from the input tree to the **Cache Return** transform, and connect the **Cache Return** transform to one or more output elements. You can then use the nested map to create or transform values and map them to the output elements that you connected to the **Cache Return** transform.

You can also add a **Cache Failure** transform to your **Cache Get** transform if you want to handle exceptions that might occur if the **Cache Get** transform fails; see "Cache Failure" on page 1290.

For more information, see "Retrieving a value from the global cache by using a Mapping node" on page 1517.

Note: The **Cache Get** transform does not have any conditional properties. To configure conditional execution, you must embed the **Cache Get** transform inside an **If** transform.

Parameters

The **Cache Get** nested mapping has a number of parameters that you configure by mapping values or assigning fixed values. The following table describes the parameters that you can set when you use the **Cache Get** transform:

Parameter	Mandatory	Description	Result if no value is configured
Key	Yes	The name of the key in the key-value pair that is contained in the global cache map.	Error
MapName	You must specify a value for this parameter if you specify a value for the **CacheName** parameter.	The name of the global cache map that contains the key-value pair.	The default map in the cache is used. If you are using the embedded cache, the default map is SYSTEM.BROKER.DEFAU
CacheName	No. But if you specify this parameter, you must also specify the **MapName** parameter.	The name of the global cache WXSServer policy to be used.	The embedded cache in the integration node is used as the cache.

Cache Put
You can use the **Cache Put** transform to insert a key-value pair into the global cache.

Overview

The **Cache Put** transform accepts only input connections, which provide values for the parameters in the nested mapping.

You can assign or map only single values to a key-value pair in the global cache. By default, the schema type of the **Value** parameter in the **Cache Put** transform is defined as `xsd:anySimpleType`. The **Value** parameter can accept any simple-typed input mapping to create a cache entry value. The cache entry is created with a Java Object type that is determined by the schema type of the mapped input element; see "Java Object types that are created for each schema input type" on page 1295. To ensure that the mapped input element has a specific schema type, and hence that the cache entry is always created with a specific Java Object type, you can apply a mapping cast to the **Value** parameter and map the input element to the resulting target element; see "Casting elements in a message map" on page 1398. Alternatively, you can use an `xs:type` transform to cast the mapped input element, or assigned value; see "Cast type (xs:type)" on page 1299.

If the element can contain multiple values, set the Cardinality property to ensure that a single value is mapped to the key or value; see "Configuring the cardinality of a user-defined element" on page 1412.

The key-value pair is stored in a map that is in the global cache.

You can choose which of the following actions the **Cache Put** transform takes by selecting from the following options on the General tab of the **Properties** pane for the **Cache Put** transform:

Insert an entry in cache
A new key-value pair is added to the global cache. If the cache already contains an entry that has the same key, an error is produced. This option is the default option.

Update an entry in cache
The value that is associated with an existing key in the global cache is updated. If the cache does not already contain an entry that has the same key, an error is produced.

Insert or update an entry in cache
The global cache is queried to determine if the cache contains an entry that has the key. If the cache already contains an entry that has the same key, the value that is associated with the key is updated in the cache. If the cache does not already contain an entry that has the same key, a new key-value pair is added to the cache.

Note: The operation might fail if the key is being concurrently added or deleted.

You can add a **Cache Return** transform to your **Cache Put** transform group if you want to return values after a successful **Cache Put** transform; see "Cache Return" on page 1297.

You can add a **Cache Failure** transform to your **Cache Put** transform group if you want to handle exceptions that might occur if the **Cache Put** transform fails; see "Cache Failure" on page 1290.

For more information, see "Adding key-value pairs to the global cache by using a Mapping node" on page 1515 .

Note: The **Cache Put** transform does not have any conditional properties. To configure conditional execution, you must embed the **Cache Put** transform inside an **If** transform.

Parameters

The **Cache Put** nested mapping has a number of parameters that you configure by mapping values or assigning fixed values. The following table describes the parameters that you can set when you use the **Cache Put** transform:

Parameter	Mandatory	Description	Result if no value is configured
Value	Yes	The value in the key-value pair that is inserted into the global cache map. For information about the Java Object types that are created for each schema input type, see “Java Object types that are created for each schema input type” on page 1295.	Error
Key	Yes	The name of the key in the key-value pair that is inserted into the global cache map. You use this key name to retrieve the associated value by using the **Cache Get** transform.	Error
MapName	You must specify a value for this parameter if you specify a value for the **CacheName** parameter or the **TimeToLive** parameter.	The name of the global cache map to receive the key-value pair. If the cache map does not exist, the map is created before the key-value pair is added.	The default map in the cache is used. If you are using the embedded cache, the default map is SYSTEM.BROKER.DEFAU
CacheName	No. But if you specify this parameter, you must also specify the **MapName** parameter, and you must not specify the **TimeToLive** parameter.	The name of the global cache WXSServer policy to be used.	The embedded cache in the integration node is used as the cache.
TimeToLive	No. But if you specify this parameter, you must also specify the **MapName** parameter, and you must not specify the **CacheName** parameter.	The time (in seconds) until the key-value pair is removed from the global cache. Enter `0` or `-1` to configure the key-value pair to remain in the cache until the integration node is restarted, or the cache is cleared by using the **mqsicacheadmin** command; see mqsicacheadmin command. **Note:** The cache ignores negative values other than -1.	The cache entry does not expire.

Note: You can set these parameters by using user-defined properties. Then, you can provide values for the user-defined properties dynamically by using a BAR file override. For more information, see “Accessing user-defined properties from a Mapping node” on page 1479.

Java Object types that are created for each schema input type
When you map an input element to a **Cache Put `Value`** parameter, the type of Java object that is created in the global cache is determined by the `http://www.w3.org/2001/XMLSchema` schema input.

When you map an input element to the **`Value`** parameter, you are prompted to apply a mapping cast to redefine the default `xs:anySimpleType` schema type to a specific schema type (see "Casting elements in a message map" on page 1398), so that the global cache entry is created with a specific Java Object type. If you do not use a mapping cast on the **`Value`** parameter, the Java Object type that is created in the global cache might vary depending on the type of the mapped input element.

The following table details the Java Object type that is created in the global cache for each type of `http://www.w3.org/2001/XMLSchema` schema input.

Note: Regardless of the type of the input element in the **Cache Put** nested mapping, if the input element has a nil value and you do not specifically cast the target parameter to a concrete type, the entry that is created in the global cache is a null value.

Type of the input element that is mapped to the Cache Put `Value` parameter	Type of Java object that is created in global cache
boolean	java.lang.Boolean
byte	java.lang.Byte
decimal	java.math.BigDecimal
double	java.lang.Double
float	java.lang.Float
int	java.lang.Integer
unSignedshort	java.lang.Integer
integer	java.lang.BigInteger
negativeInteger	java.lang.BigInteger
nonNegativeInteger	java.lang.BigInteger
nonPositiveInteger	java.lang.BigInteger
postiveInteger	java.lang.BigInteger
unsignedLong	java.lang.BigInteger
long	java.lang.Long
unsignedInt	java.lang.Long
short	java.lang.Short
unsignedByte	java.lang.Short
string	java.lang.String
date	java.lang.String (XML formatted string)
dateTime	java.lang.String (XML formatted string)
gDay	java.lang.String (XML formatted string)
gMonth	java.lang.String (XML formatted string)

Type of the input element that is mapped to the Cache Put Value parameter	Type of Java object that is created in global cache
gMonthDay	java.lang.String (XML formatted string)
gYear	java.lang.String (XML formatted string)
gYearMonth	java.lang.String (XML formatted string)
time	java.lang.String (XML formatted string)
dayTimeDuration	java.lang.String (XML formatted string)
duration	java.lang.String (XML formatted string)
yearMonthDuration	java.lang.String (XML formatted string)
anyUri	java.lang.String
base64Binary	java.lang.String
hexBinary	java.lang.String
normalizedString	java.lang.String
qName	java.lang.String

Cache Remove
You can use the **Cache Remove** transform to remove a key-value pair from the global cache.

Overview

The **Cache Remove** transform accepts only input connections, which provide values for the parameters in the nested mapping.

You can assign or map only single values of type simple value to the key. If a mapped element is not of the appropriate data type, you can use the Cast type (**xs:type**) transform to assign the data type. If the element can contain multiple values, set the Cardinality property to ensure that a single value is mapped to the key; see “Configuring the cardinality of a user-defined element” on page 1412.

You can add a **Cache Return** transform to your **Cache Remove** transform group if you want to perform some mapping after a successful **Cache Remove** transform. The **Cache Remove** transform provides the value of the entry that has been removed. You can then map this value in the **Cache Return** transform; see “Cache Return” on page 1297.

You can add a **Cache Failure** transform to your **Cache Remove** transform group if you want to handle exceptions that might occur if the **Cache Remove** transform fails; see “Cache Failure” on page 1290.

For more information, see “Removing a key-value pair from the global cache by using a Mapping node” on page 1519.

Note: The **Cache Remove** transform does not have any conditional properties. To configure conditional execution, you must embed the **Cache Remove** transform inside an **If** transform.

Parameters

The **Cache Remove** nested mapping has a number of parameters that you configure by mapping values or assigning fixed values. The following table describes the parameters that you can set when you use the **Cache Remove** transform:

Parameter	Mandatory	Description	Result if no value is configured
Key	Yes	The name of the key in the key-value pair that is contained in the global cache map.	Error
MapName	You must specify a value for this parameter if you specify a value for the **CacheName** parameter.	The name of the global cache map that contains the key-value pair.	The default map in the cache is used. If you are using the embedded cache, the default map is SYSTEM.BROKER.DEFAU
CacheName	No. But if you specify this parameter, you must also specify the **MapName** parameter.	The name of the global cache WXSServer policy to be used.	The embedded cache in the integration node is used as the cache.

Cache Return
You can use the **Cache Return** transform to return values after a successful **Cache** transform.

Overview

Include a **Cache Return** transform in your **Cache Put**, **Cache Get**, or **Cache Remove** transform group to specify a nested mapping that is called if the **Cache** transform completes successfully.

A **Cache Return** transform does not perform any transformation. The **Cache Return** nested mapping provides predefined input elements that provide data from the successful cache operation. You can also map elements from the input tree to the **Cache Return** transform and connect the **Cache Return** transform to one or more output elements. You can then use the nested map to create or transform values and map them to the output elements that you connected to the **Cache Return** transform.

The predefined Value input element in the **Cache Return** nested mapping has a default schema type of `xs:anySimpleType`. If you know the type of the Java object in the cache, you can map directly to a target element that has the schema type that matches the Java Object type; see "Schema types that are returned for each supported Java Object type" on page 1298.

If the entry in the cache might have one of several Java Object types, you can apply mapping casts to the predefined Value input element. You can apply mapping casts for the schema types that match each of the Java Object types; see "Casting elements in a message map" on page 1398. Alternatively, you can use an `xs:type` transform to cast the predefined Value input element; see "Cast type (xs:type)" on page 1299. For example, if the Java Object type of the cache entry might be `java.lang.Integer` or `java.lang.String`, you would apply a mapping cast for a schema type of `xs:int`, and a mapping cast for a schema type of `xs:string`.

Predefined elements

The **Cache Return** nested mapping has a number of predefined input elements that vary depending on the associated **Cache** transform. The following table describes the predefined input elements that are included with each **Cache** transform:

Element	Cache transforms in which element in included	Description
Value	**Cache Get** **Cache Remove**	The value from the key-value pair that is associated with the key that is mapped to the **Cache** transform. For information about the schema type that is returned for each type of Java object in the cache, see "Schema types that are returned for each supported Java Object type" on page 1298. **Note:** At run time, the **Value** element does not exist in the following situations: • There was no matching entry in the cache. • (For **Cache Remove** transforms only.) A matching entry was removed from the cache but the entry had a null value. You can test for the existence of the **Value** element by using the XPath expression `fn:exists($Value)` in the Condition tab of the transform that maps the **Value** element to the output element.
Count	**Cache Put**	The number of entries added to the global cache or removed from the global cache.

Schema types that are returned for each supported Java Object type

When you retrieve an entry from the global cache, the type of the Java object in the cache determines the `http://www.w3.org/2001/XMLSchema` schema type of the returned Value element.

The following table details the `http://www.w3.org/2001/XMLSchema` schema type of the Value element that is returned for each of the supported Java Object types.

Type of Java object that is in global cache	Schema type of the Value element that is returned
`java.lang.Boolean`	`boolean`
`java.lang.Byte`	`byte`
`java.math.BigDecimal`	`decimal`
`java.lang.Double`	`double`
`java.lang.Float`	`float`
`java.lang.Integer`	`int`
`java.lang.BigInteger`	`integer`
`java.lang.Long`	`long`
`java.lang.Short`	`short`
`java.lang.String`	`string`

Note: If you map the returned Value element to a target element that has a different schema type, a type cast occurs; see Casting from primitive types to primitive types. The cast might fail if the schema types

are incompatible. If the global cache entry was created by a **Cache Put** transform, the Java Object type of the cache entry is detailed in the following topic: "Java Object types that are created for each schema input type" on page 1295.

For example, if a **Cache Put** transform maps an element of schema type `xs:dateTime` to a cache entry, the cache entry is created with a Java Object type of `java.lang.String`. If you use a **Cache Get** transform to retrieve the entry from the cache, the Value element is returned with a schema type of `xs:string`. In the nested mapping for the **Cache Return** transform, you can map the returned Value element to a target element of schema type `xs:dateTime`.

Cast type (`xs:type`)
You can use an **xs:type** transform to cast the value of a simple element to a specific data type.

Overview

For example, you might want to assign a value with a specific data type to a target element that is defined as `xs:anySimpleType`.

When you use an **xs:type** transform, you can have zero or more input elements mapped to the transform, but only a single output element. You can cast one value to set the output element. This value is set in the **Value** column of the **General** tab on the Properties page. The value can be one of the following entities:

- A literal value
- A single input value
- An XPath expression that uses zero or more of the input values

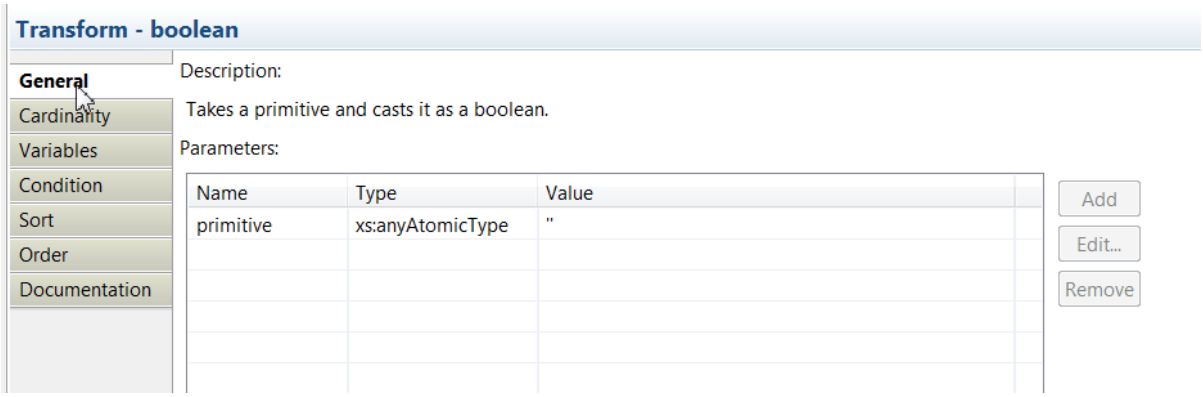
You can also use any of the input elements to build an XPath conditional expression that determines whether the **xs:type** transform is applied or not. You set this conditional expression on the **Condition** tab of the Properties page.

You must choose the **xs:type** transform according to the output element data type. For example, if you have an output element with a Boolean data type, you must choose **xs:boolean** transform.

For more information about casting a specific value type to an output element, see "Setting the value of an output element with a explicit data type" on page 1483.

Assign a value

You can set a fixed value or define an XPath expression in the **Value** field that is located in the **General** tab of a **xs:type** transform properties view.



To define an XPath expression, you click **Edit**. Then, you can use content-assist to enter the expression. As part of the expression, you can use any input elements for which you define connections to the transform.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **xs:type** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

When you define multiple input connections, you can configure the following properties to define how those inputs are displayed in the message map:

- **Sort**: You can sort the inputs to the transform by ascending order, descending order, case order, or data order.
- **Order**: You can display the order of input connections to a transform. You can reorder them.

You can use the input connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Note: You only can sort and order inputs that are connected with a primary connection to the transform.

`xs:type` transforms

You can use any of the following `xs:any` transforms:

- **xs:NOTATION**: This function takes a primitive and casts it as notation.
- **xs:Qname**: This function takes a primitive and casts it as a qualified name.
- **xs:anyURI**: This function takes a primitive and casts it as anyURI.
- **xs:base64Binary**: This function takes a primitive and casts it as base64Binary.
- **xs:boolean**: This function takes a primitive and casts it as boolean.

  You can use any of the following values: `true` Or `false`.
- **xs:dateTime**: This function takes a primitive and casts it as dateTime.
- **xs:date**: This function takes a primitive and casts it as date.
- **xs:dayTimeDuration**: This function takes a primitive and casts it as dayTimeduration.
- **xs:decimal**: This function takes a primitive and casts it as decimal.
- **xs:double**: This function takes a primitive and casts it as double.
- **xs:float**: This function takes a primitive and casts it as float.
- **xs:gDay**: This function takes a primitive and casts it as gDay.
- **xs:gMonthDay**: This function takes a primitive and casts it as gMonthDay.
- **xs:gMonth**: This function takes a primitive and casts it as gMonth.
- **xs:gYearMonth**: This function takes a primitive and casts it as gYearMonth.
- **xs:gYear**: This function takes a primitive and casts it as gYear.
- **xs:hexBinary**: This function takes a primitive and casts it as hexBinary.
- **xs:integer**: This function takes a primitive and casts it as integer.
- **xs:int**: This function takes a primitive and casts it as signed 32-bit integer.
- **xs:string**: This function takes a primitive and casts it as string.

- **xs:time**: This function takes a primitive and casts it as time.
- **xs:yearMonthDuration**: This function takes a primitive and casts it as yearMonthDuration.

For more information, see Casting from primitive types to primitive types.

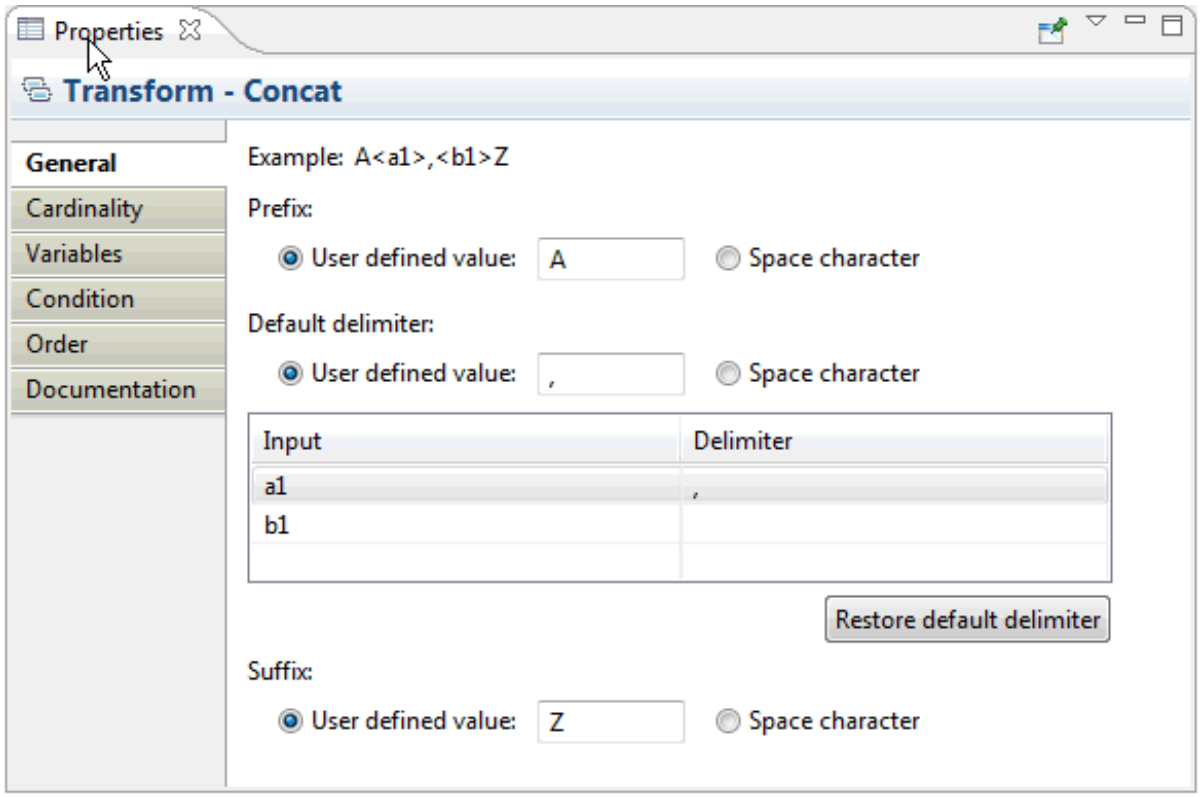
Concat
You can use a **Concat** transform to concatenate data from two or more simple elements into a string output element.

Overview

The **Concat** transform concatenates two or more simple inputs into a string output element.

When you configure the **Concat** transform, you can specify a prefix, a suffix, and a delimiter through the Properties page:

You can specify an alphanumeric character to be the delimiter between the strings,. You can also use a string prefix and a string suffix.



For example, you can concatenate the strings from the elements `firstname` and `lastname`, and specify a space as the delimiter, a prefix of `Mr. `, and a comma as the suffix, with the following result: `Mr. firstname lastname,`

When can you use the Concat transform?

You can use the **Concat** transform when the following premises apply:

- You want to concatenate data from two or more single type inputs.
- The input types to the **Concat** transform can be any simple or primitive data types.

  Note: Simple type input elements that are not of type **xs:string** will be cast to **xs:string**.
- All the inputs to the **Concat** transform, that are connected as primary connections, are used to calculate the value of the output string element.

- You might need to define a prefix.
- You might need to define the same delimiter between input values.
- You might need to define a suffix.

You cannot select and use the **Concat** transform when any of the following criteria on inputs applies:

- One of the inputs to the **Concat** transform is a complex type element.
- One of the inputs is a repeating element, that is, the input element cardinality is set to [1..*] or [0..*].

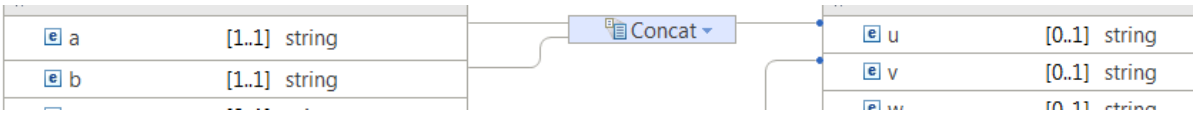
Inputs to the Concat transform

The **Concat** transform takes multiple simple type elements.

You wire inputs to the **Concat** transform as primary connections.

You can wire the same input into a **Concat** transform more than once.

The **Concat** transform concatenates input data in order. The order of inputs into the transform is recognized, and is set in the **Order** property page.



Order of the inputs

By default, the order of the inputs to the **Concat** transform is the order in which you wire the inputs.

You can modify the order by reordering the inputs in the **Order** tab of the transform properties.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Concat** transform. You can then use one or more of these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Note: You only can use inputs that are connected with a primary connection to the transform, and all inputs will be concatenated. It is not possible to have an input that is only used in the condition.

Note: When you need to apply the **Concat** transform conditionally, use an **If** transform and place the **Concat** transform within the nested mapping of the **If**.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

If your conditional expression requires the value of an input element that is not used in the **Concat** transform to set the output element, you must use the **If** transform to define when the transform is applied at run time. For more information, see “If, Else if, and Else” on page 1331.

Warning error

By default, you cannot connect a repeating simple element to a **Concat** transform. However, if you have a map where you have defined a **Concat** transform, and you change the cardinality of one of the input elements so it becomes a repeatable simple type, the **Concat** transform will show a warning.

The warning message is the following:

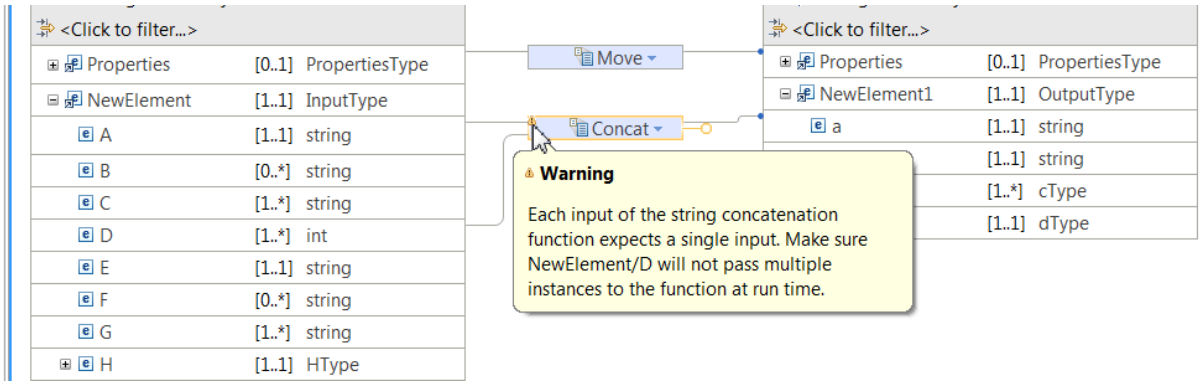
```
Each input of the string concatenation function expects a single input. Make
sure {0} will not pass multiple instances to the function at run time.
```

The warning is displayed because at least one input is not of single type.

When a **Concat** transform has such a warning, the run time behavior is the following:

- If the input XML has no more than one instance of the repeatable input, the **Concat** transform produces the expected result at run time.
- If the input XML has more than one instance of the repeatable input, the **Concat** transform results in a run time exception.

Example for a repeating single input element to a Concat transform:



Convert
The **Convert** transform implements a schema type cast on the input to match the output. If the type cast cannot be performed on the input instance value, an exception is thrown and the map processing stops.

Overview

Use the **Convert** transform to move a simple input element to an output element, when both elements are defined in the relevant schema models with different types.

You can only connect one input to the **Convert** transform. You must use a primary connection to connect the input element to the transform.

For example, you might have an input element with type `xsd:int` and an output element with type `xsd:decimal`. The **Convert** transform changes the input with an `int` type to an output with a `decimal` type.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Convert** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

You can use the input connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Create
You can use the **Create** transform to initialize an output element, set an output element to nil, or set an output element to fixed value defined by the schema.

Overview

For simple types, you cannot use the value of an input element to set the value of an output element in a **Create** transform.

However, you can define zero or more **supplement** connections between input elements and the **Create** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied.

You can use the **Create** transform to create the following types of output element:

- An empty element. For more information, see “Initializing a simple or complex output element by using the Create transform” on page 1491.
- An element with `xsi:nil="true"`, also called a nil element. For more information, see “Choosing an XPath conditional expression that tests for a nil value in a transform ” on page 1438.
- A simple type element with a default value that is specified by the schema. For more information, see “Setting the value of a simple output element to a default or fixed value” on page 1485.
- A complex element with child elements populated by using the **Assign** transform or the **Move** transform from supplemental wired inputs.

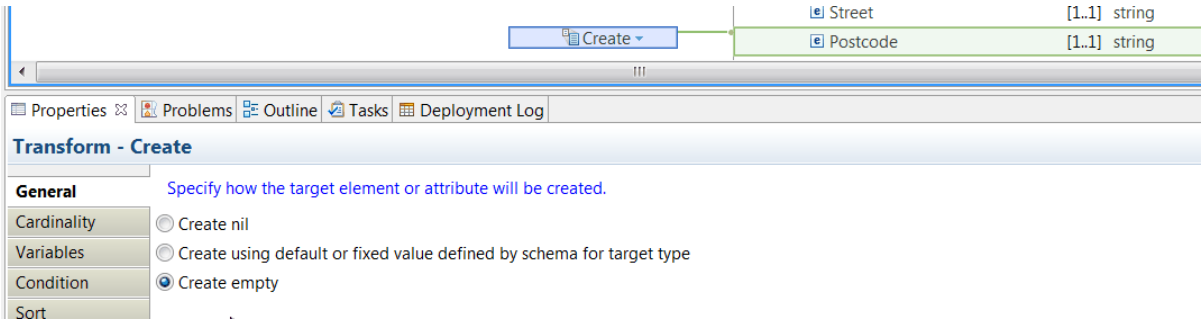
Set the value of a string type output element

By default, the option **Create empty** is preselected. You must choose this option to initialize a single type output element.

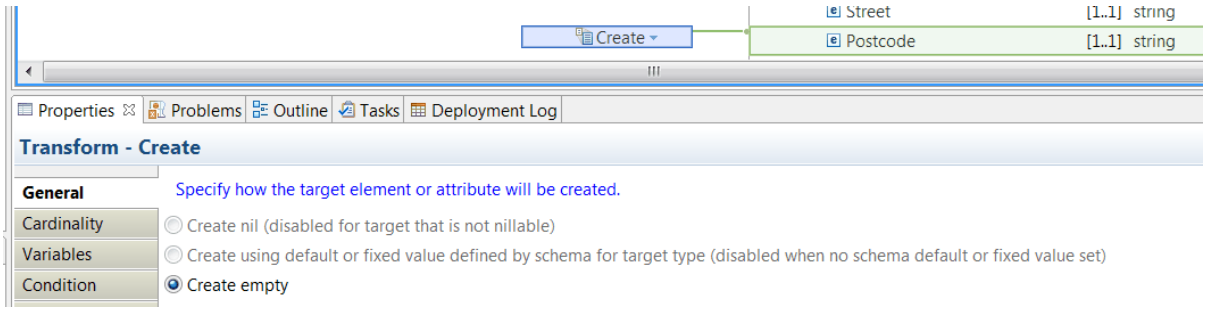
For example, for an element that is defined in the schema model as `<element name="Postcode" type="string" nillable="true" default="30567" />`, you can choose any of the following options:

- Create nil
- Create by using a default or a fixed value that is defined by schema for target type
- Create empty

The following figure shows the properties view for the **Create** transform defined for that element:



For example, if the output element is defined in the schema model as `<element name="Postcode" type="string" nillable="false" />`, you can choose to create the output element as empty.



The option to create a nil element is available only when the target is nillable.

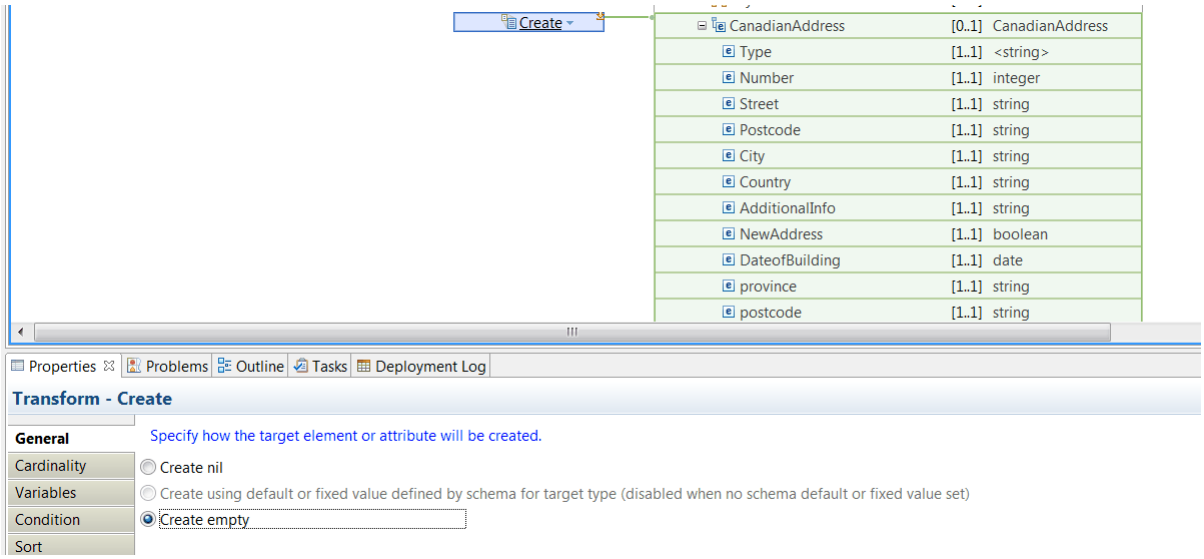
The option to create an element with the default value is only available when the target has a default value that is defined in the schema file.

Set the value of a complex type output element

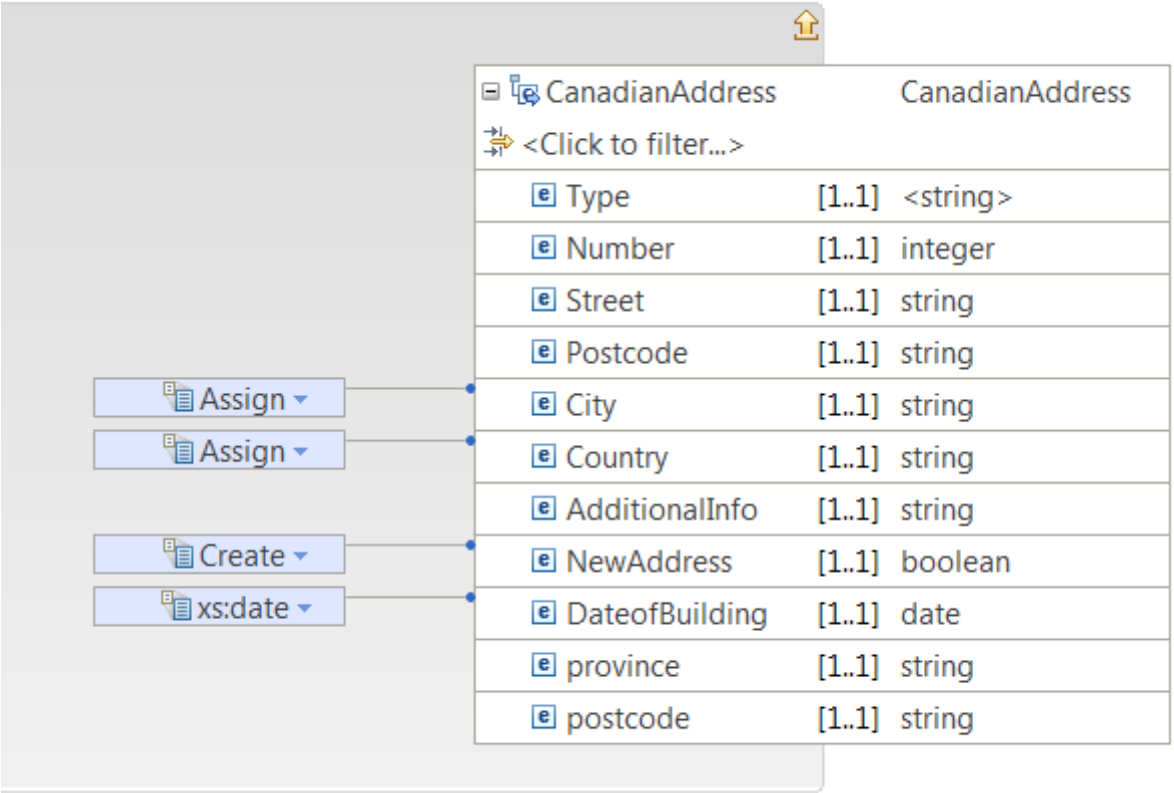
By default, the option **Create empty** is preselected. You must choose this option to initialize a complex output element.

Note: You can initialize or set to nil complex output elements.

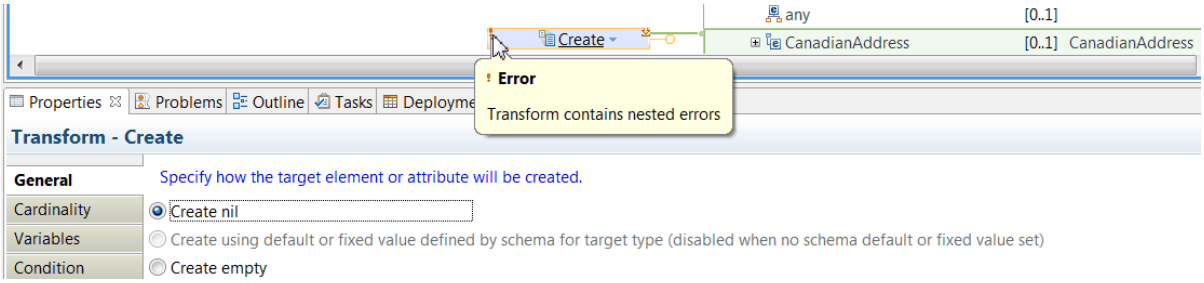
When you define a **Create** transform to initialize a complex type output element, a nested map is created containing the complex structure elements. You configure each element within the nested map individually, that is, you initialize, set to nil, or set to a default value each individual element.



When you initialize a complex type, you can use the nested map that is associated with the **Create** transform to provide values for the child element. You can define and configure each element by using transforms that do not require an input, for example the **Assign** transform, an **xs:type** transform, or the **Create** transform. You can also wire a supplemental input to move input values to the child elements.



When you define a **Create** transform to set to nil a complex type output element, a nested map is created containing the complex structure elements. You can get the error `Transform contains nested errors` if you try to configure any element within the nested map individually.



Specify how a simple data type element or attribute is created

You must specify how an output element or an attribute is created in the **General** tab of the **Create** transform properties view.

The following table lists simple data types with their default options preselected when you define a **Create** transform. It lists the options that you can choose to set the value of an output element of that type:

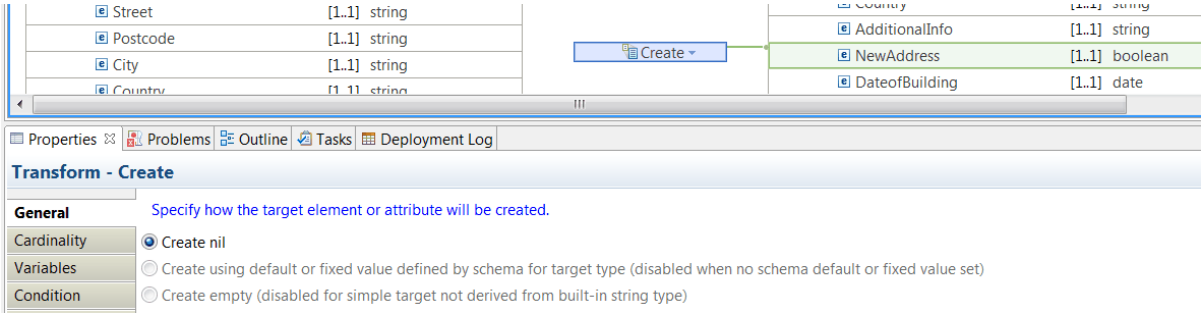
*Table 50. Configuration choices available when you define a **Create** transform to set the value of an output element*

Data type	Default option	Output element can be set to empty	Output element can be set to nil (the element is defined as *nillable="true"*)	Output element can be set to fixed or default value (the *default* property is set in the schema)
String	Create empty	valid	valid	valid
Integer	Create nil	not valid	valid	valid
Boolean	Create nil	not valid	valid	valid
date	Create nil	not valid	valid	valid
dateTime	Create using default or fixed value defined by schema for target type	not valid	not valid	valid
double	Create using default or fixed value defined by schema for target type	not valid	not valid	valid
float	Create using default or fixed value defined by schema for target type	not valid	not valid	valid
hexBinary	Create empty	valid	valid	valid
int	Create using default or fixed value defined by schema for target type	not valid	not valid	valid
time	Create using default or fixed value defined by schema for target type	not valid	not valid	valid

For example, for the following schema definition of an output element:

```
<element name="NewAddress" type="boolean" nillable="true" />
```

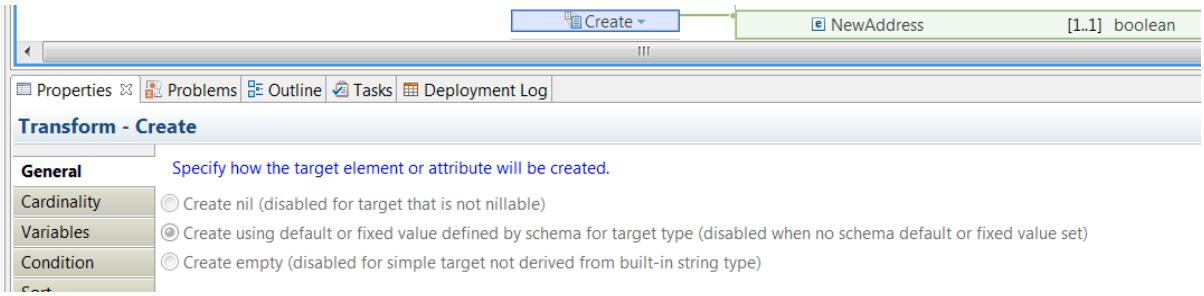
When you define a **Create** transform to set the value of a Boolean element, you can create the element as nil.



For example, for the following schema definition of an output element:

```
<element name="NewAddress" type="boolean" nillable="false" />
```

When you define a **Create** transform to set the value of a Boolean element, you can specify a fixed value, and the choice is preselected.



Define when the transform is applied at run time

You can define multiple connections between input elements and the **Create** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Note: You can use only inputs that are connected with a **supplement** connection to the transform.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

Custom ESQL
You use the Custom ESQL transform to call your own ESQL code from a graphical data map.

In the Graphical Data Mapping editor, select **Custom ESQL** from the **Custom Transforms** list. You can then use the transform properties to select ESQL code that is stored in your workspace. When you select the ESQL route, the Parameters table **Name** and **Type** columns are populated. You must then select an input element or XPath expression in the **Value** column for each parameter. You can use the content assist in the **Value** column to help you to assign the required element, literal, or XPath expression.

The following topics contain further information about ESQL types and functions:

- “Equivalent ESQL types and schema types” on page 1309
- “Equivalent ESQL and XPath mapping functions” on page 1311

The ESQL file that contains the referenced ESQL module must be visible for mapping to be selectable. Ensure that the application, library, or project references are set to make the ESQL file accessible to the map. When you deploy the map, ensure that the ESQL file is also deployed, and that **Compile and in-line resources** is not selected.

Requirements for ESQL modules that are called from a graphical data map

The following requirements apply to ESQL modules that are called from a graphical data map:

- The syntax of an ESQL procedure is shown in CREATE PROCEDURE statement. The procedures that can be called from a Custom ESQL transform in a graphical data map must conform to the following requirements:
  - Only IN parameters are allowed
  - A RETURN is required
- A return data type must be a simple scalar.
- An input parameter data type must be a simple scalar, or an ESQL REFERENCE where the reference variable meets the following criteria:
  - The reference variable is used to access only the input element or descendants of the input element.
  - The input element and any descendants that are accessed by using the reference variable are defined in the input message model.
  - The input element must exist. If the input element does not exist, the map fails and reports that the ESQL procedure input is the wrong data type.

    Note:

    If there is a chance that the input element might not exist, you can prevent the ESQL procedure from being called by adding a condition that uses `fn:exists(`*`inputElementVariable`*`)`.
- An ESQL module with no inputs can be used to assign to an output element.
- Each input parameter to the ESQL module can be taken from an input element that is wired into the custom ESQL transform or specified as a constant.
- The ESQL must not include SQL calls to a data source. The Graphical Data Mapping editor provides facilities to include database operations in the map. For more information, see "Mapping database content" on page 1523.

Note: The target element of a Custom ESQL transform is always created and given the value that is returned by the associated ESQL procedure. If the ESQL procedure returns ESQL NULL, the transform creates an element with an empty value. If you do not want an element to be created, you can add a conditional expression.

Equivalent ESQL types and schema types

When you need to process the data to set a target value, you can implement a function in Java or ESQL. If you choose to use ESQL functions, you can invoke them from your map in a custom ESQL transform.

The following table shows the equivalence between schema types and ESQL types:

XML Schema simple type	ESQL data type in message tree
anyURI	CHARACTER
base64Binary	BLOB
boolean	BOOLEAN
byte	INTEGER
date	DATE
dateTime	TIMESTAMP

XML Schema simple type	ESQL data type in message tree
dayTimeDuration	INTERVAL
decimal	DECIMAL
double	FLOAT
duration	INTERVAL
ENTITIES	Repeating set of elements in the tree, each of type CHARACTER
ENTITY	CHARACTER
float	FLOAT
gDay	DATE
gMonth	DATE
gMonthDay	DATE
gYear	DATE
gYearMonth	DATE
hexBinary	BLOB
ID	CHARACTER
IDREF	CHARACTER
IDREFS	Repeating set of elements in the tree, each of type CHARACTER
int	INTEGER
integer	DECIMAL
language	CHARACTER
long	INTEGER
Name	CHARACTER
NCName	CHARACTER
negativeInteger	DECIMAL
NMTOKEN	CHARACTER
NMTOKENS	Repeating set of elements in the tree, each of type CHARACTER
nonNegativeInteger	DECIMAL
nonPositiveInteger	DECIMAL
normalizedString	CHARACTER
NOTATION	CHARACTER
positiveInteger	DECIMAL
QName	CHARACTER
short	INTEGER
string	CHARACTER

XML Schema simple type	ESQL data type in message tree
time	TIME
token	CHARACTER
unsignedByte	INTEGER
unsignedInt	INTEGER
unsignedLong	DECIMAL
unsignedShort	INTEGER
yearMonthDuration	INTERVAL

Equivalent ESQL and XPath mapping functions
You can implement some data mapping functions either by using XPath functions or by supplying equivalent ESQL functions in a Custom ESQL transform.

The following functions in the ESQL language have equivalent XPath functions built into the Graphical Data Mapping editor. You can invoke these functions directly without having to write ESQL modules in an ESQL file to be called from the map:

Table 51. Equivalent ESQL and XPath functions

ESQL function	XPath function
EXTRACT YEAR FROM	fn:year-from-date fn:year-from-dateTime
EXTRACT MONTH FROM	fn:month-from-date fn:month-from-dateTime
EXTRACT DAY FROM	fn:day-from-date fn:day-from-dateTime
EXTRACT HOUR FROM	fn:hours-from-dateTime fn:hours-from-duration fn:hours-from-time
EXTRACT MINUTE FROM	fn:minutes-from-dateTime fn:minutes-from-duration fn:minutes-from-time
EXTRACT SECOND FROM	fn:seconds-from-dateTime fn:seconds-from-duration fn:seconds-from-time
EXTRACT DAYS FROM	fn:days-from-duration
EXTRACT MONTHS FROM	fn:months-from-duration
CURRENT_DATE	fn:current-date
CURRENT_TIME	fn:current-time
CURRENT_TIMESTAMP	fn:current-dateTime

Table 51. Equivalent ESQL and XPath functions (continued)	
ESQL function	**XPath function**
LOCAL_TIMEZONE	fn:implicit-timezone
ABS (ABSVAL)	fn:abs
CEIL (CEILING)	fn:ceiling
FLOOR	fn:floor
LEFT	fn:substring
CONTAINS	fn:contains
ENDSWITH	fn:ends-with
LENGTH	fn:string-length
LOWER (LCASE)	fn:lower-case
REPLACE	fn:replace
RIGHT	fn:substring
STARTSWITH	fn:starts-with
SUBSTRING ... FROM	fn:substring
SUBSTRING ... BEFORE	fn:substring-before
SUBSTRING ... AFTER	fn:substring-after
SUBSTRING ... FROM ... FOR	fn:substring(fn:substring(...), $for)
SUBSTRING ... BEFORE ... FOR	fn:substring(fn:substring-before(...), $for)
SUBSTRING ... AFTER ... FOR	fn:substring(fn:substring-after(...), $for)
TRANSLATE	fn:translate
UPPER (UCASE)	fn:upper-case
FIELDNAME	fn:local-name
FIELDNAMESPACE	fn:namespace-uri
CARDINALITY	fn:count
EXISTS	fn:exists

Custom Java
In the Graphical Data Mapping editor, you can use the Custom Java transform to enter Java code in a message map.

Overview

You can use a Custom Java transform to process elements that have a simple type, and the type matches a native Java type.

You can use a Custom Java transform to process input and outputs that are arrays or complex types. For more information, see "Processing complex or repeating elements in a Custom Java transform" on page 1498.

Guidelines to design a Java method to transform simple types

The Java class that you provide to the map must have a static method that returns the appropriate type for the value of the output element, and takes parameters of the appropriate type for the wired inputs.

For example, the following Java method can be used in a Custom Java transform that has three input elements, of types `xs:string`, `xs:decimal`, and `xs:boolean`, and the output element has a `xs:decimal` type:

```
public static BigDecimal calSomething(String memType, BigDecimal stdCost, boolean flag) {
 BigDecimal actualCost = stdCost;
 if (flag & memType.startsWith("gold")) {
  BigDecimal discRate = new BigDecimal(0.9);
  actualCost = actualCost.multiply(discRate);
 }
 return actualCost;
}
```

Guidelines to design a Java method to transform complex or repeating types

You can use a Custom Java transform to process input and outputs that are arrays or complex types. The IBM App Connect Enterprise `Java plugin API MbElement Java class` passes data only in a Custom Java Transform. The `Java DOM API` passes data in Custom Java Transforms and in Transform condition expressions.

You can use the MbElement class in a Java method to pass a complex type, a wildcard **xsd:any**, or a wildcard **xsd:anySimpleType**.

Make a Java class visible

To make a Java class visible in the transform, ensure that the Java project the class is in is referenced by the project that contains the map.

1. In the **Application development** page, right-click the project where the map is defined.
2. Select **Managed included projects**.
3. Select the Java project, and then click **OK**.

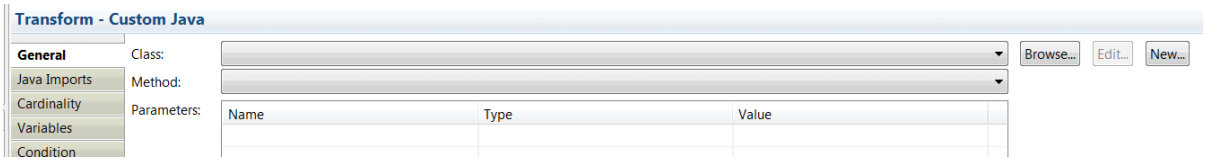
Note: Notice that if the Java project that contains the Java class does not build in Eclipse, then the Java class is not visible anywhere in the map. You must resolve all the Java errors, which you can see in the **Problems** tab, before you can configure the **Java imports** tab in the **Properties** page of the map.

Call a Java method

You can only configure one call to a Java method in a Custom Java transform. You can define more operations (java method or XPath expression) on the parameters that are required by the method.

You can configure the Java class, and a Java method in the **General** tab of the **Properties** page of the transform. The method that you specify defines the transformation logic that is applied by the Custom Java transform.

Note: The Java class that contains the method must be available in a Java project in your workspace. The Java class must be visible to the project that contains the map.



When you add a Java call in the **General** tab of the **Properties** page of a **Custom Java** transform in your map, an import is automatically added in the map to refer to the package qualified Java class, and a prefix

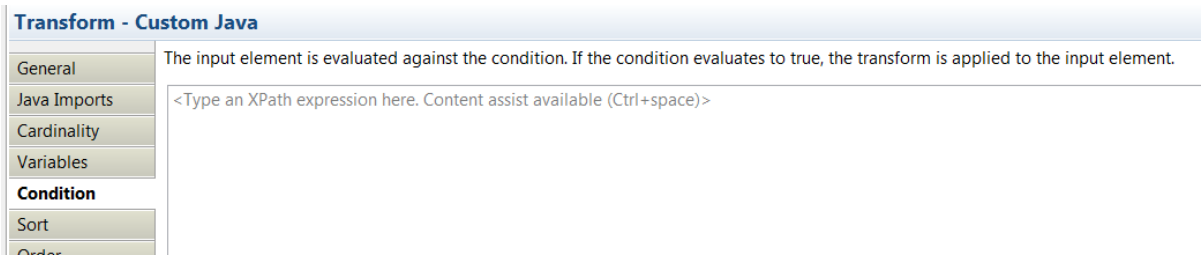
based on the class name is added. All the **public static methods** in this Java class are then available in content assist when building expressions in other transforms.

For more information, see "Mapping an element by using a Custom Java transform" on page 1496.

Define when the transform is applied at run time

You can define **supplement** connections between input elements and the Custom Java transform. You can then use any input element in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

To configure the conditional expression, you define an XPath expression or a Java expression in the **Condition** tab that is available in the **Properties** page of the transform.



To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression that includes XPath, Java, and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

By default, if the Java method does not provide a value, the transform creates an element with an empty value. If you do not want an element to be created, you can add a conditional expression in the **Condition** tab.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433, and "Defining a Java conditional expression for a transform" on page 1439.

Mappings between the schema type, the Java type, and the IBM App Connect Enterprise message assembly type

The following table shows the mappings between the schema type, the Java type, and the IBM App Connect Enterprise message assembly type:

Table 52. Mapping the schema type, Java type, and message tree element type

Schema type	Java type	IBM App Connect Enterprise message assembly tree element type
xs:anyURI	java.lang.String	CHARACTER
xs:base64Binary	byte[]	BLOB
xs:boolean	boolean, java.lang.Boolean	BOOLEAN
xs:byte	byte, java.lang.Byte	INTEGER
xs:date	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:dateTime	javax.xml.datatype.XMLGregorianCalendar	TIMESTAMP
xs:dayTimeDuration	javax.xml.datatype.Duration	INTERVAL

Table 52. Mapping the schema type, Java type, and message tree element type (continued)		
Schema type	**Java type**	**IBM App Connect Enterprise message assembly tree element type**
xs:decimal	java.math.BigDecimal	DECIMAL
xs:double	double, java.lang.Double	FLOAT
xs:duration	javax.xml.datatype.Duration	INTERVAL
xs:float	float, java.lang.Float	FLOAT
xs:gDay	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:gMonth	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:gMonthDay	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:gYear	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:gYearMonth	javax.xml.datatype.XMLGregorianCalendar	DATE
xs:hexBinary	byte[]	BLOB
xs:int	int, java.lang.Integer	INTEGER
xs:integer	java.math.BigInteger	DECIMAL
xs:long	long, java.lang.Long	INTEGER
xs:negativeInteger	java.math.BigInteger	DECIMAL
xs:nonNegativeInteger	java.math.BigInteger	DECIMAL
xs:nonPositiveInteger	java.math.BigInteger	DECIMAL
xs:normalizedString	java.lang.String	CHARACTER
xs:positiveInteger	java.math.BigInteger	DECIMAL
xs:short	short, java.lang.Short	INTEGER
xs:string	java.lang.String	CHARACTER
xs:time	javax.xml.datatype.XMLGregorianCalendar	TIME
xs:unsignedByte	Short	INTEGER
xs:unsignedInt	Long	INTEGER
xs:unsignedLong	java.math.BigInteger	DECIMAL
xs:unsignedShort	Int	INTEGER
xs:yearMonthDuration	javax.xml.datatype.Duration	INTERVAL

Custom XPath
You can use the **Custom XPath** transform to provide a data value for a simple target element, or values for a repeating simple target element by using an XPath expression.

Overview

You define the XPath expression in the **General** tab of the Properties view.

You can use context assist by pressing **Ctrl+Space** while constructing the XPath expression. For more information, see “Using content assist (Mapping syntax)” on page 1443.

When to use the Custom XPath transform

You can use the **Custom Xpath** transform to perform any of the following mappings:

- Implement an arithmetic operation, such as add, subtract, or multiply. For more information, see "Choosing a transform to perform an arithmetic operation" on page 1366.
- Implement complex XPath expressions. You can extend built-in transformation functions by using custom XPath expressions. For more information about XPath, see XPath tutorial or W3C XML Path Language (XPath) 2.0.
- To access user-defined properties. For more information, see "Accessing user-defined properties from a Mapping node" on page 1479.
- To set an output to null or nill, use `iib:nullValue()`. For more information, see "Creating a nil output element" on page 1487.
- To set an output hex binary BLOB value, use `iib:hexBinaryValue($<var>`
- To set an output base64 binary value, use `iib:base64BinaryValue( $<var>`
- To set a new UUID value, use `iib:uuidValue()`

Inputs

Inputs are optional to **Custom XPath** transforms. If required, you can connect any input elements to a Custom XPath transform and access them as part of the XPath expression.

Outputs

You can use a **Custom XPath** transform to set the data value of a simple target element, or to set the values for a repeating simple target element.

By default, if the XPath expression does not provide a value, the transform creates the target element with an empty value. If you do not want the target element to be created, you can add a conditional expression in the Condition tab.

Note: The **Custom XPath** transform returns values, not elements. You cannot use a **Custom XPath** transform to create elements other than the simple target element. For this reason, you cannot populate a complex structure from a **Custom XPath** transform.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Custom XPath** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

You can use any inputs that are connected with primary connections to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression that comprises XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

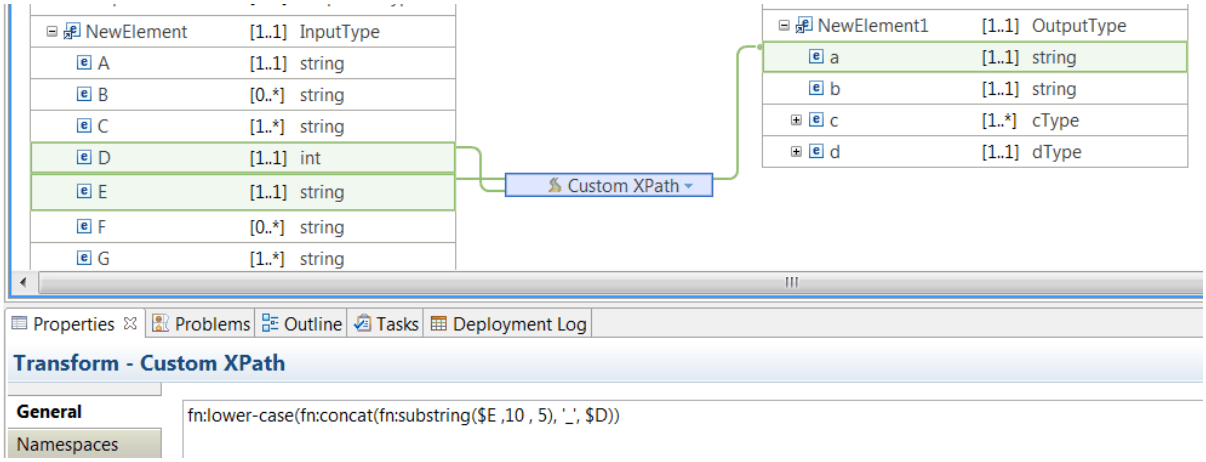
Example

The following figure shows a **Custom XPath** transform that performs the following calculation:

1. Uses the **fn:substring** XPath function to obtain part of the information provided in element **E**.
2. Uses the **fn:concat** XPath function to concatenate the result of the **fn:substring** function, with the delimiter **_**, and the element **D**.
3. Uses the **fn:lower-case** XPath function to put in lower case the result of the **fn:concat** function.

The XPath expression is a complex expression where you nest multiple XPath functions to perform the calculation. The XPath expression is the following:

```
fn:lower-case(fn:concat(fn:substring($E,10,5),'_',$D))
```



If you input the following message:

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>A1</A>
  <B>B1</B>
  <C>Field_1</C>
  <D>1000</D>
  <E>CUSTOMER_AREA1</E>
  <F>Optional1</F>
  <G>Optional1</G>
</NewElement>
```

you get the following output value:

```
<NewElement1>
 <a>area1_1000</a>
</NewElement1>
```

Example: How to perform an arithmetic operation

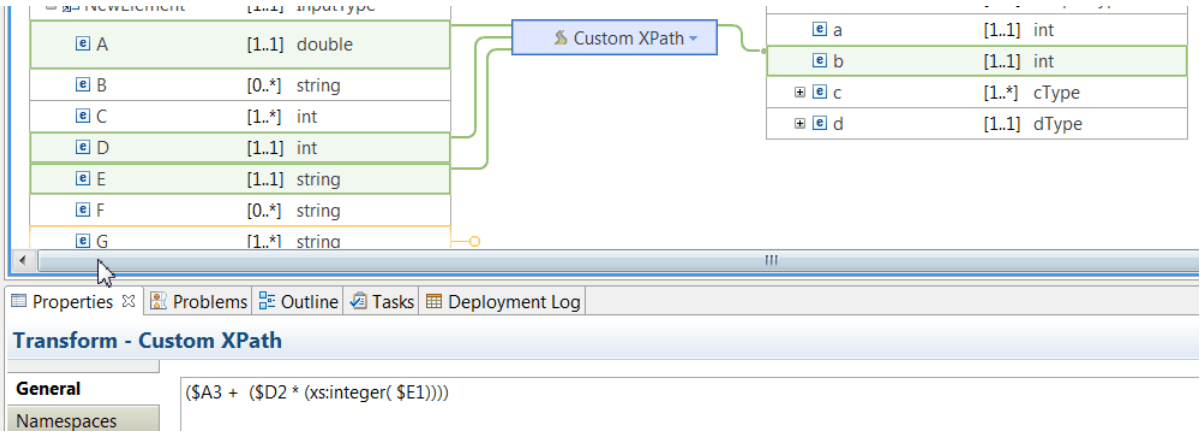
This example shows how to perform an arithmetic operation by using a **Custom XPath** transform.

The transform has 3 inputs, that are used as part of the calculation that determines the value of the output element **b**.

Note: You can only use as arguments of an arithmetic operation non-repeating simple type input elements.

The operations consists on adding the value of the input element **A** to the value of calculating **D** times **E**. The value of **E** is of type string. The value of **E** is cast to an integer.

```
($A3 + ($D2 * (xs:integer($E1))))
```



When you transform the following message, the value of **b** is **120010**.

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>10</A>
  <C>2</C>
  <C>10</C>
  <D>1000</D>
  <E>120</E>
  <F>Optional1</F>
  <G>Optional1</G>
</NewElement>
```

Database Routine
You can use the Database Routine transform to call a stored procedure in a database, setting the parameter values by mapping input elements.

The definition of the stored procedure used during the creation of your Database Routine transform is specified in a database definition file (`.dbm` file) that you select from the Data Design project.

You create the information that is provided in the database definition file by running discovery against your database server. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475. The information available varies depending on the vendor of your database server. Some information, such as the content of any Result Sets returned, is not provided. If you want to map data from any columns in these Result sets, you must define them in the Database Routine setup.

For more information, see "Calling a stored procedure from a map" on page 1532.

Limitations

Note: For information about the support for stored procedures, see "Support for stored procedures" on page 1318.

Support for stored procedures
When you call stored procedures from a map, you must be aware of the following limitations:

- Stored procedures from only the following databases are supported:
  - IBM DB2
  - Oracle

- The following entities are not supported in database calls from a graphical data map:
  - Stored procedure parameters of type **Array**
  - Database user-defined functions
  - Oracle stored procedures that are contained within packages
  - Oracle stored procedures that include IN or INOUT SYS_REFCURSOR parameters
- Stored procedures that return result sets that contain user-specified column names (column names that do not exist in any table) can be called, but you cannot include these column names in mappings.

Delete
The **Delete** transform deletes one or more rows of data from a database table.

Use the **Delete** transform to delete one or more rows of data that match a configured **Where** clause from a database table. For more information, see "Deleting data from a table" on page 1528.

The database table structure needed to create your Delete transform is specified in a database definition file (`.dbm` file) that you select from the Data Design project used to create the message map. The database definition can be discovered by connecting to a data server from the tooling.

When your message map has been deployed to an integration server, and your message map is run, the database server that is used by the IBM App Connect Enterprise runtime component to process your Delete transform is specified by the JDBC Providers policy. The JDBC Providers policy name must have the same name as the database name that is specified in your Delete transform during development.

Failure
Use the **Failure** transform to handle exceptions that might be raised by the configured database server when your message map runs SQL statements to implement the action of a database transform.

If you want the message map to handle exceptions that are returned from the database server when the SQL operation is run, instead of such exceptions stopping the message map and being reported, you can add a **Failure** transform to the transform group.

The **Failure** transform is an optional transform that can be added or removed as required.

The **Failure** transform does not perform any transformation. You must transform the input and output elements within the nested map.

Handling database warnings

When you create a database transform, select **Treat warning as error** if you want the database warnings to be treated as errors.

If this option is selected, the first SQL operation that results in a warning from the selected database raises an exception.

Run time behavior

The way that database exceptions are handled is determined by the configuration of the corresponding **Failure** transform in a message map:

1. If the **Failure** transform is present in the message map, and is connected to one or more output objects, then the exception is caught and processed by the **Failure** transform.
2. If the **Failure** transform is present in the message map, but is not connected, then the exception is caught by the **Failure** transform, and is ignored.
3. If the **Failure** transform has been deleted from the message map, then the exception is handled by the Mapping node in your message flow, and is handled in the same way as other message flow exceptions.
4. If you want the message map to stop running when the database transform receives an SQL exception, remove the **Failure** transform. For more information, see "Handling database exceptions in a graphical data map" on page 1535.

For Each

You can use the **For Each** transform to iterate over one input array element, which can be either a simple type or a complex type repeating structure, and set the value of an output element.

Overview

The **For Each** transform contains a nested map. You must map the elements in the nested map, otherwise no action is executed when the transform runs.

You can configure the **For Each** transform to run when there are input elements that match your input selection criteria.

You can configure the **For Each** transform to execute once when the input array is empty by setting the **Allow Empty input** condition in the **Filter Inputs** property tab.

You can specify the indexes of the input array that participate in the **For Each** operation by setting the **Cardinality** property.

You can specify an XPath condition over the indexes that participate in the **For Each** operation by setting the **Filter Inputs** property. This condition determines which indexes are applied.

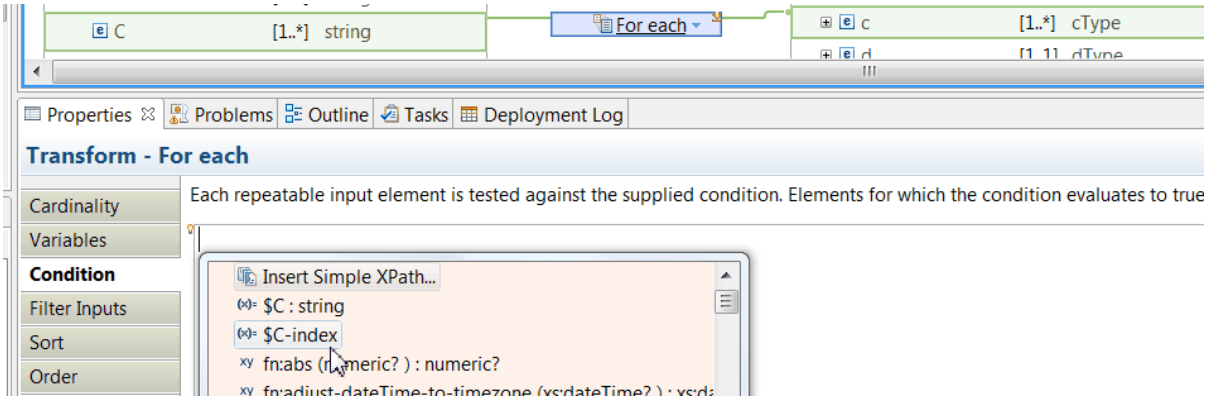
Note: The XPath condition is tested against each instance of the input array, after you apply the **Cardinality** property.

Inputs

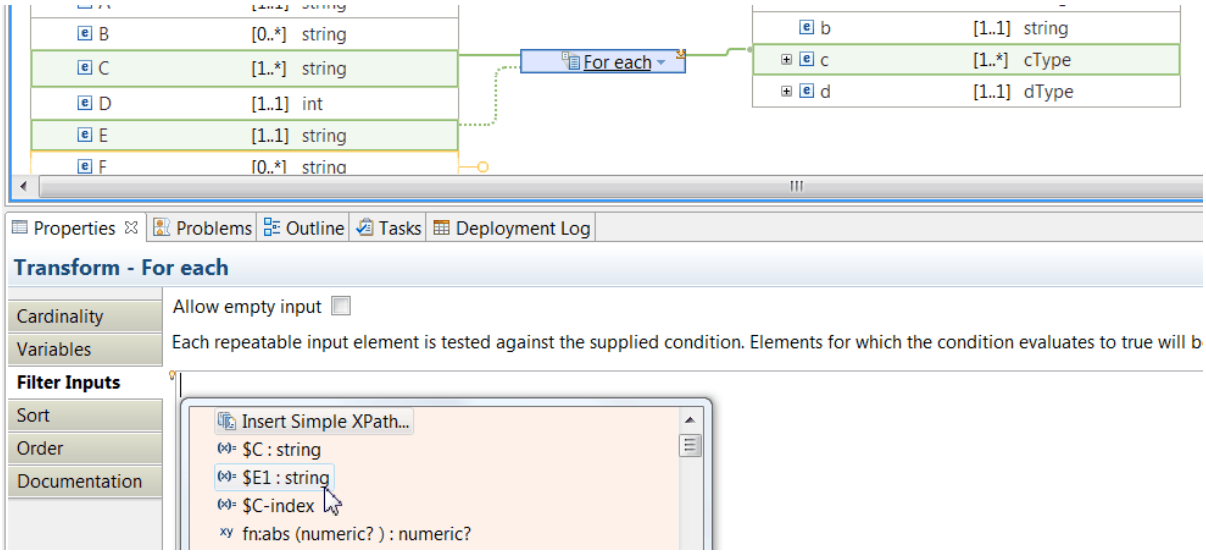
You can define multiple inputs to a **For Each** transform.

- The **For Each** transform can have only one **primary** input connection, and the input must be repeatable.
  - The Graphical Data Mapping editor counts the indexes from **1** to **N** for each processed input.
  - When the input array element is empty or no inputs match a provided filter condition, the Graphical Data Mapping editor sets the index variable to **0**.
  - The Graphical Data Mapping editor provides a variable that contains the index value of each iteration of the **For Each** transform. The name is as follows: *$VarName-index*, where *VarName* is the name of the repeating element.

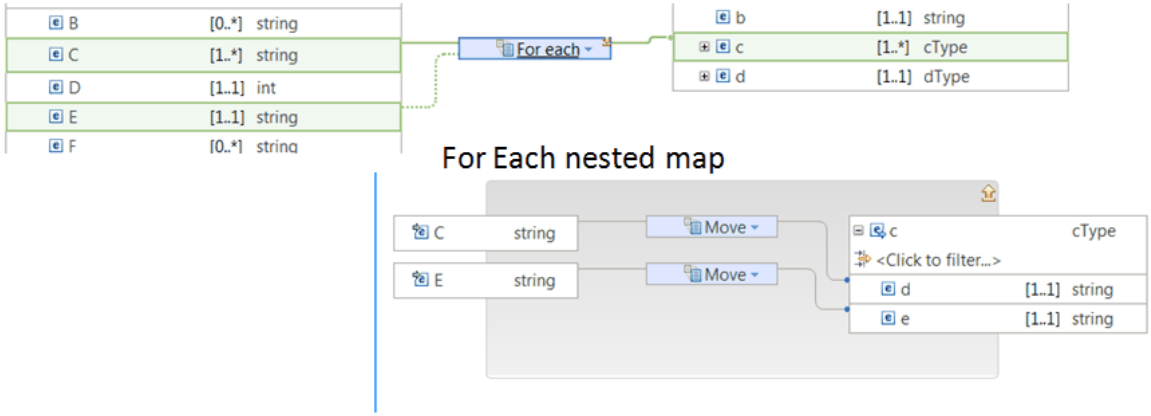
    Note: Always use content assist to get the element name assigned by the Graphical Data Mapping editor to the repeating element.



- Extra connections to the **For Each** transform must be of type **Supplement**. You can use these inputs in any of the following ways:
  - You might want to create a **Filter Inputs** expression based on the value of the input. You can use it to determine which indexes to apply as part of the transformation.



  - You might want to pass an extra element into the nested transform. This input is available in the mapping of each iteration that is run by the **For Each** transform.



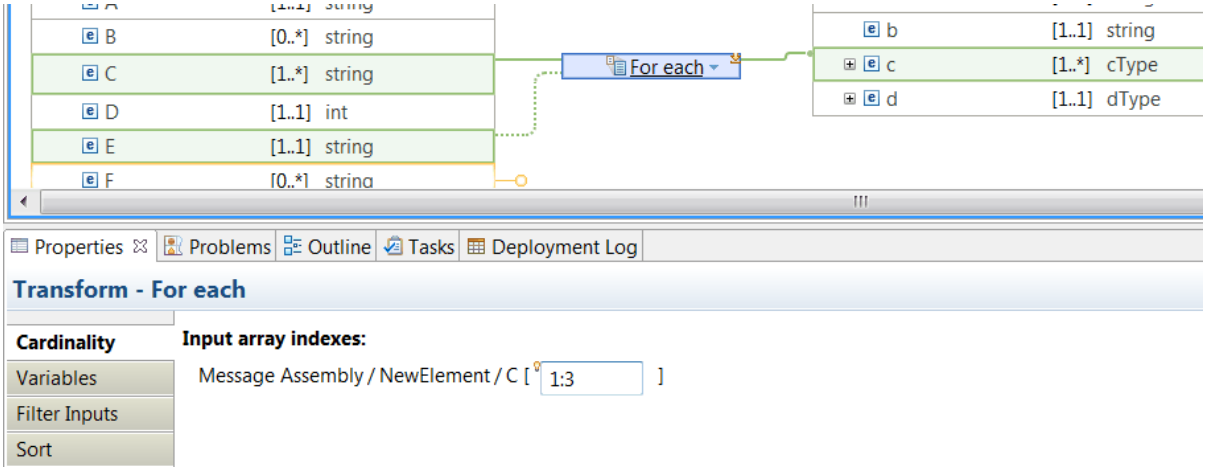
- You can configure the **Cardinality** property page on the **For Each** transform to indicate which index of an input array to iterate over.
- You can configure the **Filter Inputs** property page to specify the matching criteria for filtering input repeating elements.

Cardinality

The **Cardinality** property determines the inputs that participate in the **For Each** operation.

The first index element is 1.

You configure the **Cardinality** tab in the Properties view to specify the indexes that are processed by the transform. For more information, see “Selecting the indexes of input array elements” on page 1363.



Filter Inputs

Configure the **Filter Inputs** property tab to specify an XPath expression that determines which instances of the repeating input are processed in the nested mapping. Each element of the repeatable input is tested against the condition. The transform runs for those elements that satisfy the condition.

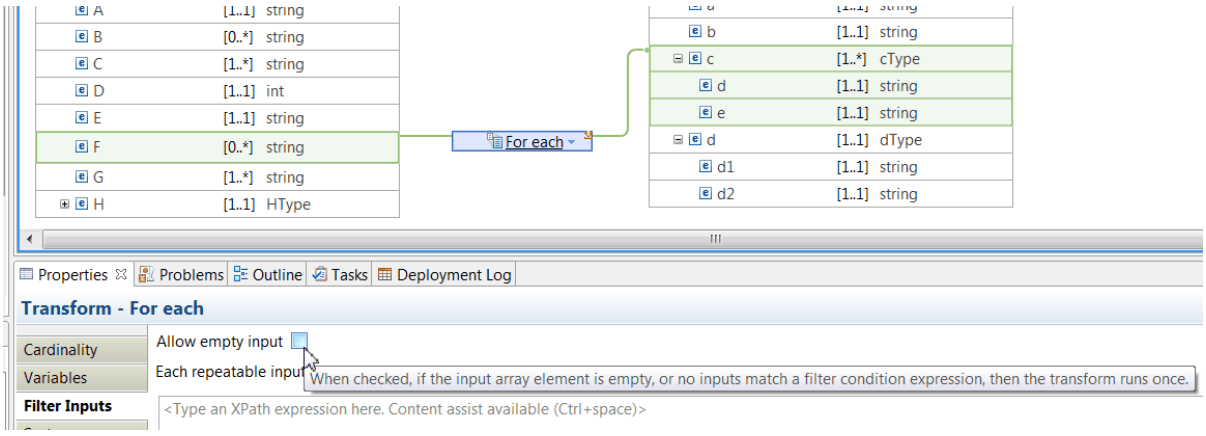
Note: The XPath condition is tested against each instance of the input array, after the **Cardinality** property is applied.

You can use XPath, or methods from Java classes to define the **Filter Inputs** expression. You can also create a complex expression comprising XPath and Java.

You can use context assist by pressing **Ctrl+Space** while you construct the **Filter Inputs** expression. For more information, see "Using content assist (Mapping syntax)" on page 1443.

Note: To obtain the exact name of the variables that are associated with the inputs, use content assist.

Enable **Allow Empty input** when the input array element is empty or no inputs match a provided filter condition, so that the transform is still entered exactly once. In this case, the primary input in the nested transform is missing, and the index variable is zero.



Outputs

The output element of a **For Each** transform can be a simple element, or a complex element, that can be a repeating element or a non-repeating element.

The output array size is equal to the input array size, minus any elements that are filtered out after applying the **Cardinality** and the **Filter Inputs** properties.

Variable $VarName-index

In the **For Each** transform, you can use the *$VarName-index* variable to indicate the index of the input array that the Graphical Data Mapping editor is processing.

Note: The *$VarName-index* variable reports indexes after the **Cardinality** property is applied.

For example, you might have an input array with 6 elements. When you set the **Cardinality** property of a **For each** transform to 1 , 3, you are telling the Graphical Data Mapping editor that you only want to process index 1 and index 3 of your input array. In this use case, the *$VarName-index* has a value of 1 for the first entry and a value of 3 for the second entry.

You can use the variable *$VarName-index* as part of the XPath condition that you can define to filter inputs, or as part of any transformation within the nested map that is associated to the **For each** transform.

The fist value of *$VarName-index* is 1 when your input array is not empty.

The value of *$VarName-index* is 0 when your input array is empty. You must enable the **Allow Empty input** property.

Variable $VarName-count

In the nested mapping of the **For Each** transform, you can use the *$VarName-count* variable to determine the number of times the Graphical Data Mapping editor enters the nested mapping to allow populating the output array.

The variable *$VarName-count* initial value is **1** . The maximum value is determined by the **Cardinality** and the **Filter Inputs** properties that are applied.

You cannot use the variable *$VarName-count* as part of the XPath condition that you can define to filter inputs.

You can use the variable *$VarName-count* as part of any transformation within the nested map that is associated to the **For each** transform.

For example, you might have an input array with 6 elements.

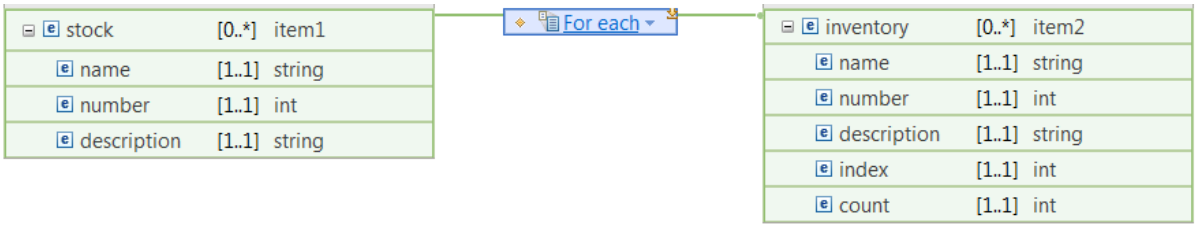
- When you set the **Cardinality** property of a **For each** transform to 1 : 3, you are telling the Graphical Data Mapping editor that you only want to process index 1, index 2 and index 3 of your input array. In this use case, the nested map is executed 3 times and the *$VarName-count* variable is equal to 1 the first time is executed, 2 the second time is executed, and then 3.
- When you have a **Filter expression** that only matches index 1 and 3 of you input array, the *$VarName-count* variable is equal to 1 the first time the nested mapping is entered for the data of input index 1. The *$VarName-count* variable is equal to 2 on the second time the nested mapping is entered for the data of input index 3.

The value of *$VarName-count* is 1 when you enable **Allow Empty input** in a **For each** transform and the nested mapping is entered as a result of the empty condition.

Example 1: Filter inputs by configuring the cardinality property

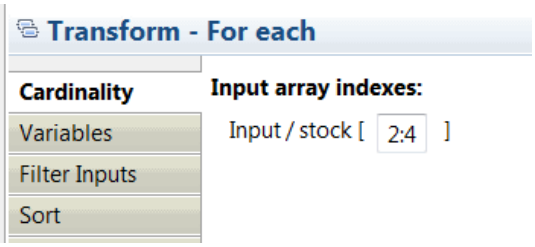
In this example, the **For Each** transform only runs for a restricted number of elements of the repeating element. The rest of the elements in the array are not considered.

The **For Each** transform iterates over instances 2, 3, and 4 between the input element **stock** and the output element **inventory**.



To restrict the number of instances, you can configure the **Cardinality** tab in the **Properties** page of the transform. You must set the following value in the **Cardinality** property tab:

```
2:4
```



When you run the following message through the **For Each** transform,

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Input xmlns:tns="http://www.example.org/schema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://www.example.org/schema foo.xsd ">
  <tns:stock>
    <tns:name>apple</tns:name>
    <tns:number>1</tns:number>
    <tns:description>fruit</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>banana</tns:name>
    <tns:number>2</tns:number>
    <tns:description>fruit</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>cap</tns:name>
    <tns:number>3</tns:number>
    <tns:description>accessory</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>door</tns:name>
    <tns:number>4</tns:number>
    <tns:description>furniture</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>elephant</tns:name>
    <tns:number>5</tns:number>
    <tns:description>animal</tns:description>
  </tns:stock>
 </tns:Input>
```

you get the following output:

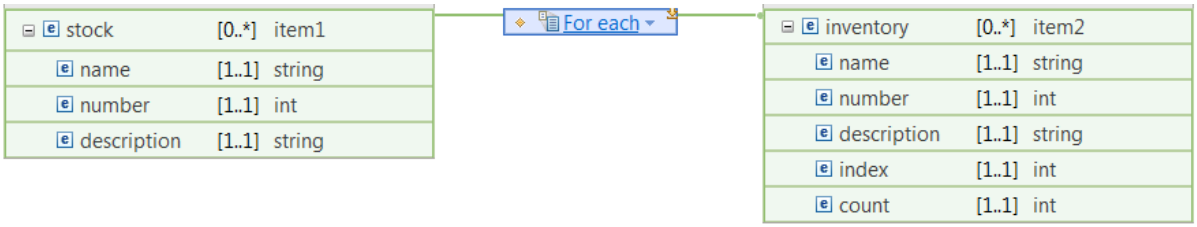
```
<?xml version="1.0" encoding="UTF-8"?><io:Output xmlns:io="http://www.example.org/schema">
  <io:inventory>
    <io:name>banana</io:name>
    <io:number>2</io:number>
    <io:description>fruit</io:description>
    <io:index>1</io:index>
    <io:count>1</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>cap</io:name>
    <io:number>3</io:number>
    <io:description>accessory</io:description>
    <io:index>2</io:index>
```

```
    <io:count>2</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>door</io:name>
    <io:number>4</io:number>
    <io:description>furniture</io:description>
    <io:index>3</io:index>
    <io:count>3</io:count>
  </io:inventory>
 </io:Output>
```

Example 2: Filter inputs by configuring the Filter Inputs property

In this example, the **For Each** transform only runs for a restricted number of elements of the repeating element. The rest of the elements in the array are not considered. The **For Each** transform iterates over instances where the name of a stock item is a string with more than 4 characters.

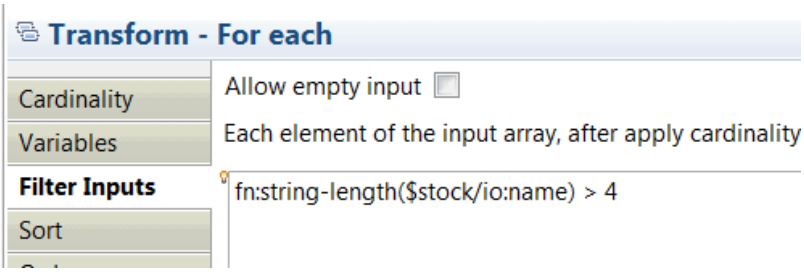
In the **For Each** transform, the input element is **stock** and the output element is **inventory**.



The **For Each** transform iterates over instances where the value of **stock.name** is a string with more than 4 characters.

To restrict the number of instances, you can configure the **Filter Inputs** tab in the **Properties** page of the transform. You must set the following value in the **Filter Inputs** property tab:

```
fn:string-length($stock/io:name) > 4
```



When you run the following message through the **For Each** transform,

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Input xmlns:tns="http://www.example.org/schema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://www.example.org/schema foo.xsd ">
  <tns:stock>
    <tns:name>apple</tns:name>
    <tns:number>1</tns:number>
    <tns:description>fruit</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>banana</tns:name>
    <tns:number>2</tns:number>
    <tns:description>fruit</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>cap</tns:name>
    <tns:number>3</tns:number>
    <tns:description>accessory</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>door</tns:name>
```

```
    <tns:number>4</tns:number>
    <tns:description>furniture</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>elephant</tns:name>
    <tns:number>5</tns:number>
    <tns:description>animal</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>flower</tns:name>
    <tns:number>6</tns:number>
    <tns:description>decoration</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>gold</tns:name>
    <tns:number>7</tns:number>
    <tns:description>money</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>honey</tns:name>
    <tns:number>8</tns:number>
    <tns:description>food</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>igloo</tns:name>
    <tns:number>9</tns:number>
    <tns:description>cold</tns:description>
  </tns:stock>
  <tns:stock>
    <tns:name>jeep</tns:name>
    <tns:number>10</tns:number>
    <tns:description>car</tns:description>
  </tns:stock>
</tns:Input>
```

you get the following output:

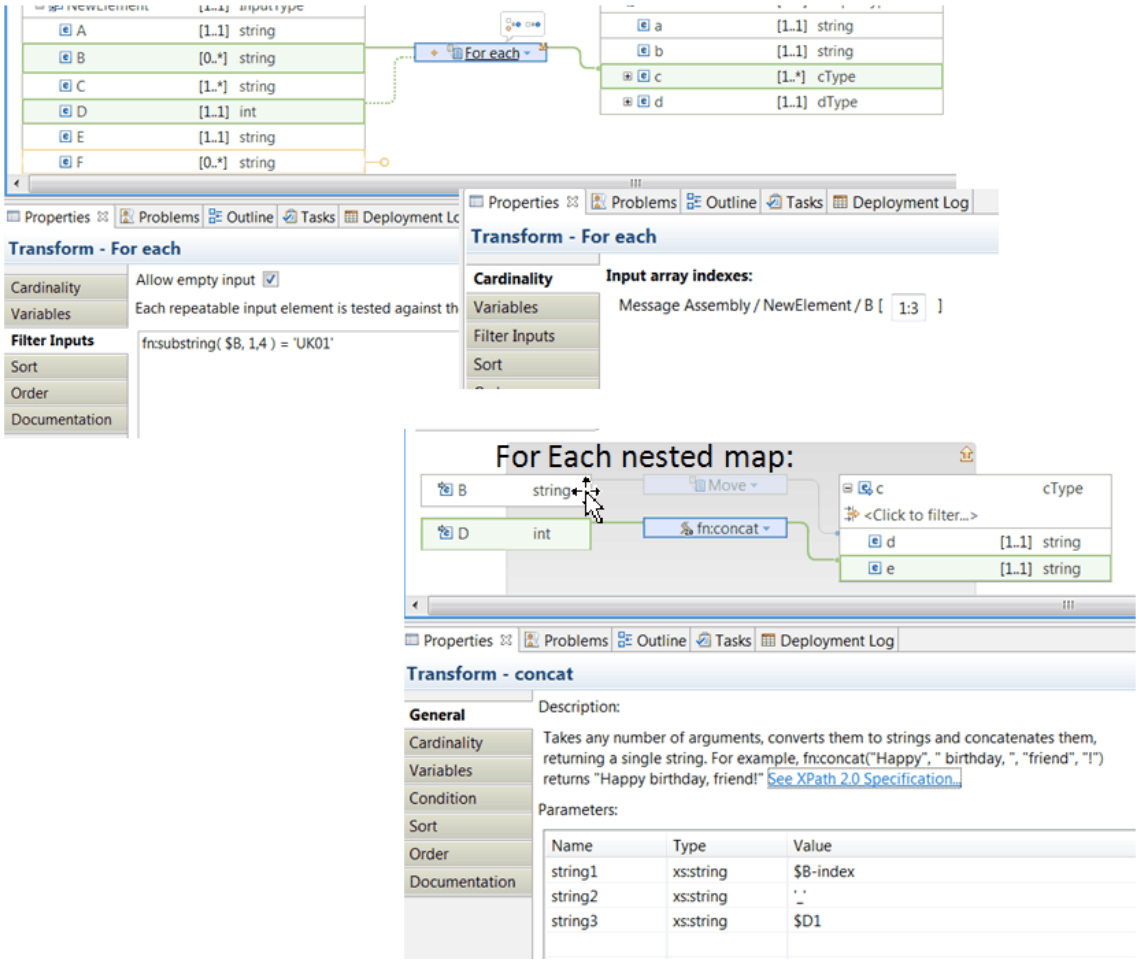
```
<?xml version="1.0" encoding="UTF-8"?><io:Output xmlns:io="http://www.example.org/schema">
  <io:inventory>
    <io:name>apple</io:name>
    <io:number>1</io:number>
    <io:description>fruit</io:description>
    <io:index>0</io:index>
    <io:count>1</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>banana</io:name>
    <io:number>2</io:number>
    <io:description>fruit</io:description>
    <io:index>1</io:index>
    <io:count>1</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>elephant</io:name>
    <io:number>5</io:number>
    <io:description>animal</io:description>
    <io:index>4</io:index>
    <io:count>2</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>flower</io:name>
    <io:number>6</io:number>
    <io:description>decoration</io:description>
    <io:index>5</io:index>
    <io:count>3</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>honey</io:name>
    <io:number>8</io:number>
    <io:description>food</io:description>
    <io:index>7</io:index>
    <io:count>4</io:count>
  </io:inventory>
  <io:inventory>
    <io:name>igloo</io:name>
    <io:number>9</io:number>
    <io:description>cold</io:description>
    <io:index>8</io:index>
    <io:count>5</io:count>
  </io:inventory>
```

```
</io:Output>
```

Example 3: Filter inputs by configuring the Cardinality and the Filter Inputs properties

In this example, the **For Each** transform only runs for the first three elements of the repeating element. The rest of the elements in the array are not considered. For each element in the array, if the first 4 characters of the element B start with UK01, then the transformation inside the nested map is executed.

Inside the nested map, the **fn:concat** transform calculates the value of element **e** based on the input element index and the input element **D**.



When you run the following message through the **For Each** transform,

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>A1</A>
  <C>Field_1</C>
  <C>Field_2</C>
  <C>Field_3</C>
  <D>1000</D>
  <E>CUSTOMER_AREA1</E>
</NewElement>
```

you get the following output:

```
<NewElement1>
 <c>
```

```
  <d/>
  <e>0_1000</e>
 </c>
</NewElement1>
```

Note: If the repeating element is empty, then the nested map is executed once because the option **Allow Empty input** is selected.

When you run the following message through the **For Each** transform,

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>A1</A>
  <B>UK011234567</B>
  <B>B2</B>
  <B>UK019999999</B>
  <B>UK01xxxxxxx</B>
  <C>Field_1</C>
  <C>Field_2</C>
  <C>Field_3</C>
  <D>1000</D>
  <E>CUSTOMER_AREA1</E>
</NewElement>
```

you get the following output:

```
<NewElement1>
 <c>
  <d>UK011234567</d>
  <e>1_1000</e>
 </c>
 <c>
  <d>UK019999999</d>
  <e>3_1000</e>
 </c>
</NewElement1>
```

Group
You can use the **Group** transform to map a single list of elements into a nested list of elements based on an input element.

Overview

For example, you can use the **Group** transform to map a single list of company employee records into a nested list of employees by department.

The **Group** transform provides a nested transform for mapping each collated input array occurrence to the output nested array structure.

The nested transforms are performed sequentially for each collated group of input elements that have matching values in the element that you have selected to group by in the **General** tab.

Inputs and outputs

The input to a **Group** transform must be a repeating complex type, such as an array or a list, and contain a simple type element that will be used for collating groups.

The output to a **Group** transform must be a repeating complex type, that includes the following elements:

- A none repeating simple type element in which to set the group identifier.
- A repeating complex type into which to map the data from the matching inputs.

Group by

In the **General** tab of the properties page, you configure the **Group by** selection.

You can select one simple type element from the input data to the **Group** transform.

The value of the simple type element is used to collate occurrences of the repeating input.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Group** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

You can use the input connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Example

You can use the **Group** transform to map a single list of company employee records into a nested list of employees by department.

The input to the **Group** transform is the repeating array **EMPLOYEE**.

The output to the **Group** transform is the repeating array **Department**.

The schema file that contains the schema for the repeating array **EMPLOYEE** is the following:

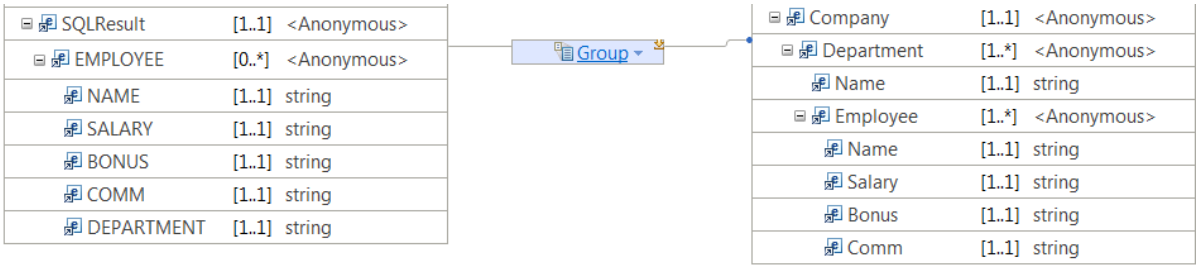
```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:employee="http://www.ibm.com"
 targetNamespace="http://www.ibm.com">
    <element name="SQLResult">
        <complexType>
            <sequence>
                <element ref="employee:EMPLOYEE" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>

    <element name="EMPLOYEE">
        <complexType>
            <sequence minOccurs="1" maxOccurs="1">
                <element ref="employee:NAME" minOccurs="1" maxOccurs="1"/>
                <element ref="employee:SALARY" minOccurs="1" maxOccurs="1"/>
                <element ref="employee:BONUS" minOccurs="1" maxOccurs="1"/>
                <element ref="employee:COMM" minOccurs="1" maxOccurs="1"/>
                <element ref="employee:DEPARTMENT" minOccurs="1" maxOccurs="1"/>
            </sequence>
        </complexType>
    </element>

    <element name="NAME" type="string"/>
    <element name="SALARY" type="string"/>
    <element name="BONUS" type="string"/>
    <element name="COMM" type="string"/>
    <element name="DEPARTMENT" type="string"/>
</schema>
```

The schema file that contains the schema for th repeating array **Department** is the following:

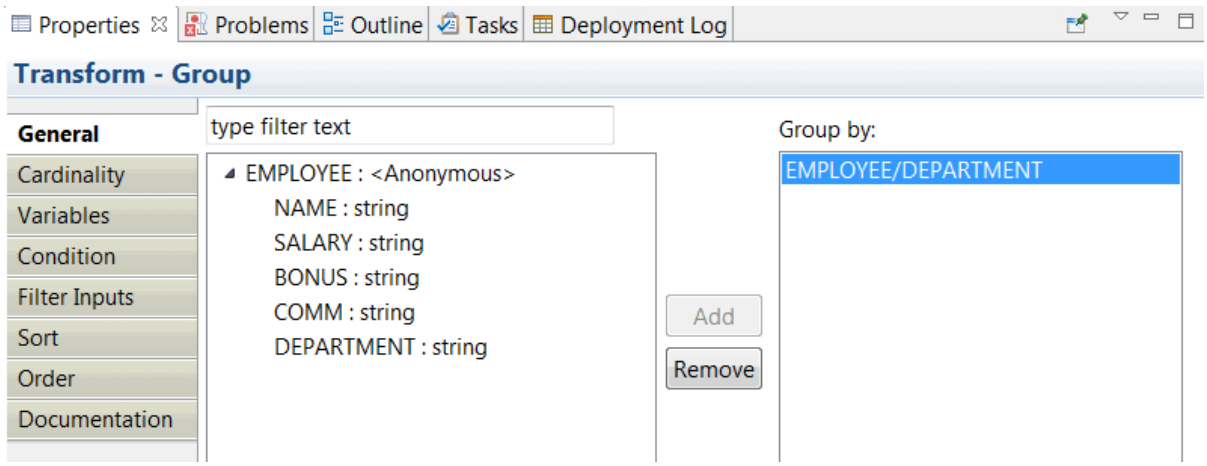
```
<?xml version="1.0"?>
```

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:se="http://www.ibm.com"
 targetNamespace="http://www.ibm.com">
    <element name="Company">
        <complexType>
            <sequence>
                <element ref="se:Department" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>

    <element name="Department">
        <complexType>
            <sequence minOccurs="1" maxOccurs="1">
                <element ref="se:Name" minOccurs="1" maxOccurs="1"/>
                <element ref="se:Employee" minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
    </element>

    <element name="Employee">
        <complexType>
            <sequence minOccurs="1" maxOccurs="1">
                <element ref="se:Name" minOccurs="1" maxOccurs="1"/>
                <element ref="se:Salary" minOccurs="1" maxOccurs="1"/>
                <element ref="se:Bonus" minOccurs="1" maxOccurs="1"/>
                <element ref="se:Comm" minOccurs="1" maxOccurs="1"/>
            </sequence>
        </complexType>
    </element>

    <element name="Name" type="string"/>
    <element name="Salary" type="string"/>
    <element name="Bonus" type="string"/>
    <element name="Comm" type="string"/>
</schema>
```

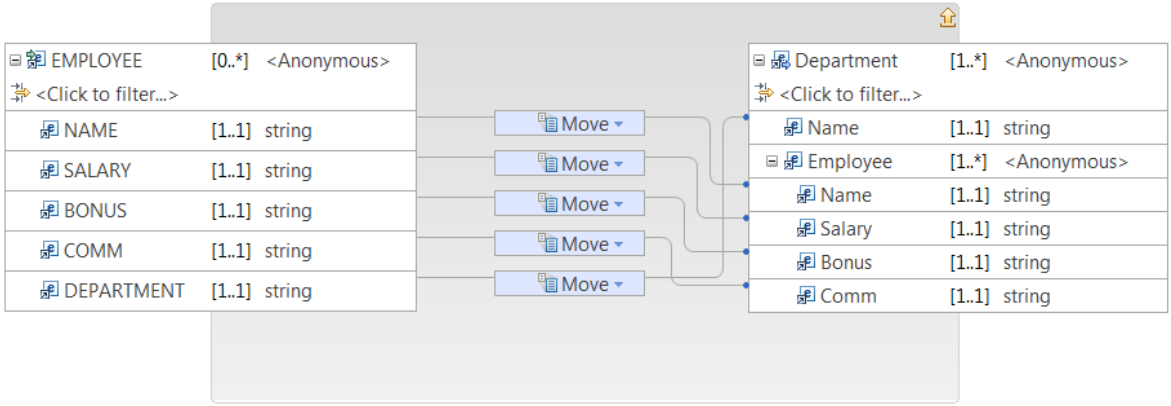
The following figure shows the map with a **Group** transform defined between **EMPLOYEE** and **Department**.



In the **General** tab of the **Properties** view for the **Group** transform, you select **DEPARTMENT:string**. Then, click **Add** to include it in the **Group by** selection. The value of **DEPARTMENT** is used to collate occurrences of **EMPLOYEE**.



Next, you open the nested map associated to the **Group** transform. You define the mappings to set the output elements of **Department**.



When you run the following input message:

```
<?xml version="1.0"?>
<employee:SQLResult xmlns:employee="http://www.ibm.com" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://www.ibm.com Employee.xsd ">
  <employee:EMPLOYEE>
    <employee:NAME>SAM HIGH</employee:NAME>
    <employee:SALARY>52750.00</employee:SALARY>
    <employee:BONUS>1000.00</employee:BONUS>
    <employee:COMM>4220.00</employee:COMM>
    <employee:DEPARTMENT>DEVELOPMENT</employee:DEPARTMENT>
  </employee:EMPLOYEE>
  <employee:EMPLOYEE>
    <employee:NAME>SALLY TANGO</employee:NAME>
    <employee:SALARY>38250.00</employee:SALARY>
    <employee:BONUS>800.00</employee:BONUS>
    <employee:COMM>3060.00</employee:COMM>
    <employee:DEPARTMENT>HUMAN RESOURCE</employee:DEPARTMENT>
  </employee:EMPLOYEE>
  <employee:EMPLOYEE>
    <employee:NAME>JOHN SMITH</employee:NAME>
    <employee:SALARY>40175.00</employee:SALARY>
    <employee:BONUS>800.00</employee:BONUS>
    <employee:COMM>3214.00</employee:COMM>
    <employee:DEPARTMENT>DEVELOPMENT</employee:DEPARTMENT>
  </employee:EMPLOYEE>
</employee:SQLResult>
```

You get the following output message:

```
<io:Company xmlns:io="http://www.ibm.com">
<io:Department>
  <io:Name>DEVELOPMENT</io:Name>
  <io:Employee><io:Name>SAM HIGH</io:Name><io:Salary>52750.00</io:Salary><io:Bonus>1000.00</
io:Bonus><io:Comm>4220.00</io:Comm></io:Employee>
  <io:Employee><io:Name>JOHN SMITH</io:Name><io:Salary>40175.00</io:Salary><io:Bonus>800.00</
io:Bonus><io:Comm>3214.00</io:Comm></io:Employee>
</io:Department>
<io:Department>
   <io:Name>HUMAN RESOURCE</io:Name>
   <io:Employee><io:Name>SALLY TANGO</io:Name><io:Salary>38250.00</io:Salary><io:Bonus>800.00</
io:Bonus><io:Comm>3060.00</io:Comm></io:Employee>
</io:Department>
</io:Company>
```

If, Else if, and Else

The **If, Else if, and Else** transform enables you to control the flow of the mapping by setting conditions.

The **If, Else if and Else** transform operates as a group of conditional transforms. The condition is applied to the input element of the conditional transform. If the condition is satisfied, the transform that is nested within the conditional transform is run.

- For each conditional transform in the group, enter a condition in the **Condition** tab in the Properties view.

  You can define a condition expression that is an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising, XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.
- To change the order in which the conditions are evaluated, select the conditional group and click the **Order** tab and use the up and down arrows.
- Double-click the conditional transform (for example, **If**) to create the mapping that will execute for the condition.

The elements in the nested map must be mapped in order for the transform to run.

Insert
The **Insert** transform inserts a row to a database table.

Use the **Insert** transform to insert a row into a database table. For more information, see "Inserting data into a table" on page 1525.

The database server that is used during the creation of your **Insert** transform is specified in a database definition file (`.dbm` file) that you select from the Data Design project used to create the message map.

When your message map has been deployed to an integration server, and your message map is run, the database server that is used by the IBM App Connect Enterprise runtime component to process your **Insert** transform is specified by the JDBC Providers policy. The JDBC Provider policy name must have the same name as the database name that is specified in your **Insert** transform during development.

Join
You can use a **Join** transform to join elements from two or more inputs into an output element.

Overview

You provide a **Join expression** to determine which data to transform.

You define the mapping from the joined instances of the input arrays to an output target in the nested map. For more information, see "Using nested maps" on page 1371.

The **Join** transform implements an inner-join.

If you want to implement an outer-join, use the **For Each** transform with the option **Allow Empty input** enabled.

Inputs

The inputs to a **Join** transform can be repeating simple elements or complex type elements, which can be merged using nested transforms to create an output.

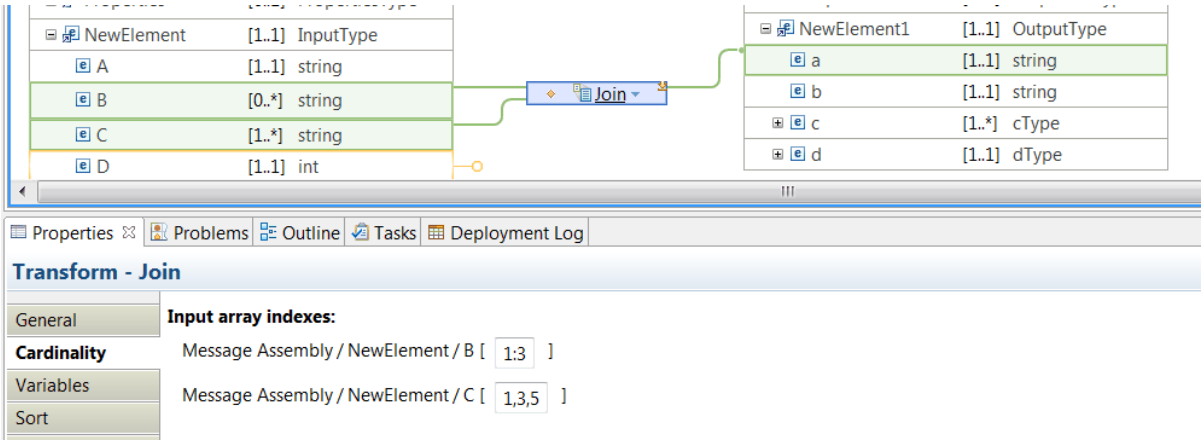
- You configure the inputs to a **Join** transform with primary connections.
- You must have at least 2 primary inputs to a **Join** transform.
- You can configure the **Cardinality** property page of each input to indicate which index of an input array to iterate over.
- You can configure the **Order** property page to define the order of iteration over the input elements.
- You can define a **Join expression** to specify the matching criteria for joining or filtering input repeating elements. The join expressions determines the size of the output element.

You can define additional inputs to the **Join** transform by using supplement connections. These inputs are passed unchanged to the nested map. You can use these inputs inside your nested map to define conditional expressions in the transforms that you define.

Cardinality

The **Cardinality** property determines the inputs participating in the join operation.

You configure the **Cardinality** property tab of a **Join** transform to define which indexes are used from each input array in the join operation. For more information, see “Selecting the indexes of input array elements” on page 1363.



Order of the inputs

By default, the order of the inputs to the **Join** transform is the order in which you wire the inputs.

You can modify the order by reordering the inputs in the **Order** tab of the transform properties.

Output

The output element can be a simple element or a complex type element that can be repeating or not repeating.

When your output element is a repeating element, the **Join expression** determines the size of the output element.

For example, if you have a repeating input element of size M, and a repeating input element of size N to a **Join** transform, the output element size is calculated as follows:

- If you click the option **Create Join expression based on index**, the size of the output element is the minimum value of M and N.
- If you do not define a **Join expression** or you do not select the option **Create Join expression based on index**, the size of the output element is *MxN*.
- If you define a **Join expression**, the size of the output element is determined by the expression.

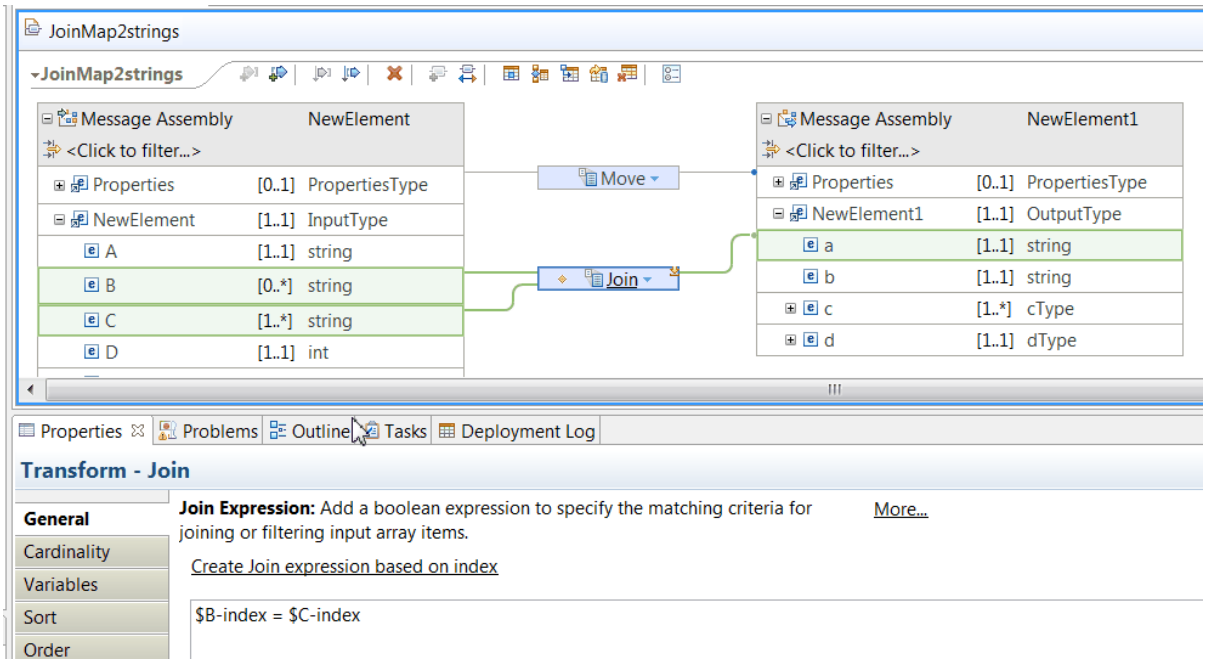
Example 1: To join the first element of the first array with the first element of the second array, you set a join condition that matches based on index. You click **Create Join expression based on index**.

Example 2: If you do not specify a **Join expression**, the join matches the first element of the first array with all elements of the second array, and then it matches the second element of the first array with all elements of the second array, and so on.

Join expression

You can define a **Join expression** to specify the matching criteria for joining or filtering input array elements. This expression is an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression comprising, XPath, Java and extension functions such as **mb:getUserDefinedProperty("propertyname")**.

- You can use content assist to create the expression. For more information, see "Using content assist (Mapping syntax)" on page 1443.
- You can select the option **Create Join expression based on index**. This option requires two repeatable inputs to create the Join expression.



For example, you can use **iib:getUserDefinedProperty("propertyname")** on a For Each filter to use a value that is not hard coded in the map, and you can also set dynamically via **CMP API**.

Local map

A **Local map** is a transform that provides a hierarchical view of element transforms in a message map. A **Local map** enables you to create a more efficient transformation because the nested mapping is executed by default if the primary input is available at run time.

Overview

You can use local maps to break up a large map into nested groups of mapping elements and process the complex elements of the whole data object.

Local maps are a partial view of a larger map, rather than separate files.

The local map does not transform data. You must specify transforms for the input and output elements in the nested map.

Inputs and outputs

A local map has at least one element as input (either a simple type or a complex type), which can contain nested elements. You can add more inputs to a local map, as supplementary, to allow extra simple type or complex type element data to be available in the nested mapping, or use them to provide a condition for when the nested mapping is executed.

The output can be either a single element or an array element, but it must be a complex type.

If you add another input as the primary input, the Graphical Data Mapping editor typically changes the transform type from **Local map** to a more applicable type. For example, when you add an extra simple element as a primary input to a local map that has an existing primary input, the Graphical Data Mapping editor changes the transform type to **Concat**.

When you connect a repeating input to a **Local map** transform, only the first instance of the array is processed. If you need to process multiple indexes of the array, you must use the **For Each** transform. For more information, see "For Each" on page 1320.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Local map** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

You can use the input connected with a primary connection to the transform. To add more inputs, or to provide a condition for when nested mapping is executed, you can define **Supplement** connections between input elements and the transform. If you do not provide a condition, and the primary input is optional in the schema model (`minOccurs=0`), the nested mapping of the local map is executed only if the primary input is available.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

Move
You can use the **Move** transform to copy data from one input element to one output element.

Overview

You can define the **Move** transform between single simple elements or between complex type elements.

You can only connect one input to the **Move** transform. You must use a primary connection to connect the input element to the transform.

You can use the **Move** transform between complex type input and output elements only if the input and output have the same type, or if the type of the input is derived from the type of output; for example, if the input element's type is USA`ddress` and the output type is A`ddress`.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Move** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

You can use the input connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

Normalize
The **Normalize** transform normalizes the input string by removing white space such as spaces, tabs, and returns, and moves the resulting normalized string to the output element.

Overview

For example, it can be used to remove multiple occurrences of white space characters before a comparison of data is done.

The **Normalize** transform is functionally equivalent to the XPath 2.0 `fn:normalize-space()` function. For more information about XPath functions, see the online document W3C XML Path Language (XPath) 2.0.

You can only connect one input to the **Normalize** transform. You must use a primary connection to connect the input element to the transform.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Normalize** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

You can use the input connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

Remove
You can use the **Remove** transform to delete an element from the Environment tree or to remove an element within an **Overrides** group.

Overview

For example, you can use the **Remove** transform to remove child elements from the target structure if you copy a complex structure from source to target.

The **Remove** transform does not have any inputs wired to the transform.

You can define a **Remove** transform on a simple, complex, or repeating output element.

The **Remove** transform is available in the list of available transforms when you define a transform on an element in the environment tree or when you define a transform on an element within an **Overrides** group.

Note: A **Remove** transform in an **Overrides** group cannot be used to remove a specific xsi:typed element; the **Remove** transform uses only the element name to identify the target element to remove.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **Remove** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

To add inputs to the transform, you can define **supplement** connections between input elements and the transform.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

Modifying repeating elements

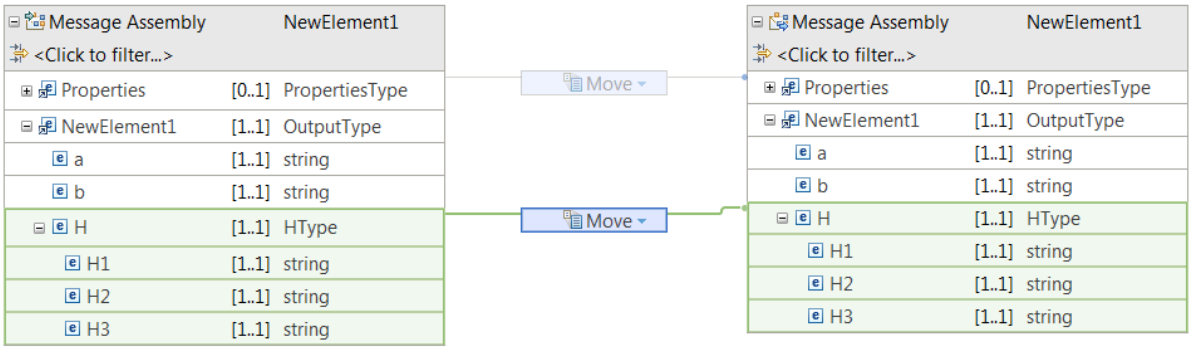
If you need to remove some elements of a repeating structure, use the **For Each** transform and define conditional expressions for each index to determine whether the element is moved or removed.

For more information, see "For Each" on page 1320.

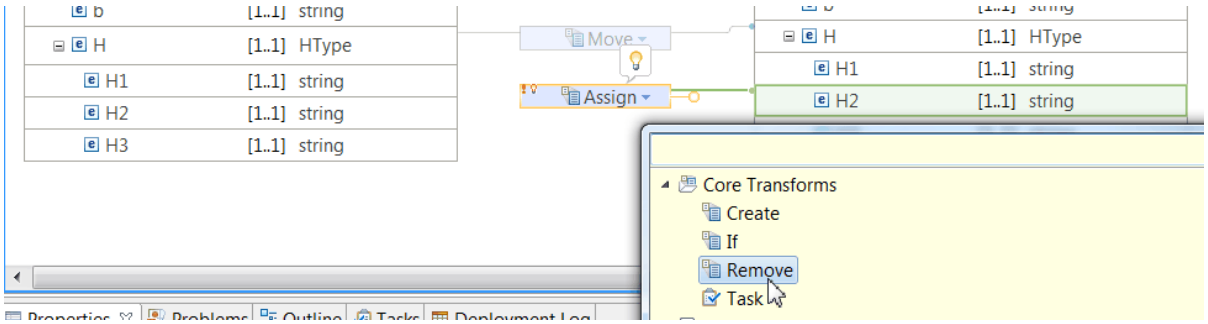
Example

This example shows how to copy an input structure to the output, and then delete one element from the output message structure.

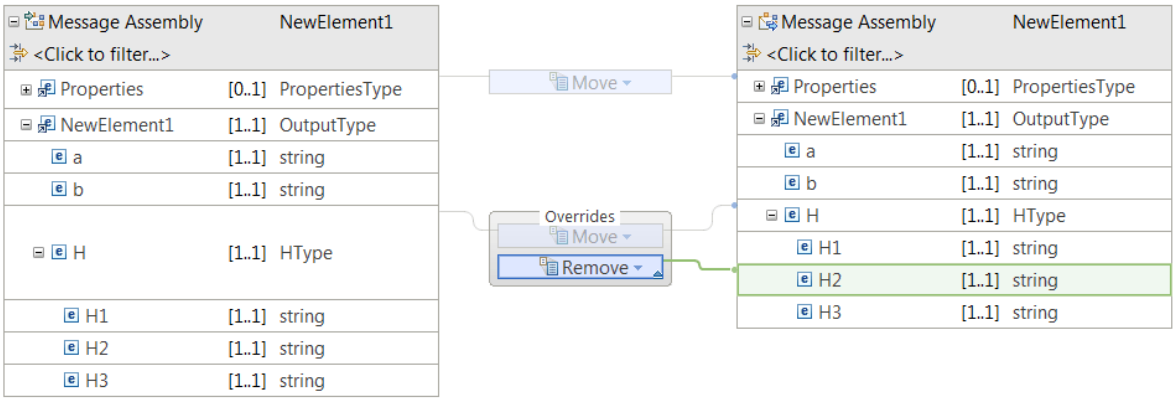
You define a map with a **Move** transform between the input element and the output element.



You click the element that you want to delete (**H2**) from the output structure, and then drag and drop the mouse onto the canvas. An **Assign** transform is added. You change the **Assign** transform to the **Remove** transform.



As soon as you select the **Remove** transform, the Graphical Data Mapping editor adds an **Overrides** function that groups the **Move** transform and the **Remove** transform. The **Overrides** function specifies that the input element **H** is copied over unchanged, and then element **H2** is deleted from the output structure.



Return

Use the **Return** transform to report the number of rows that are modified by a database transform.

The **Return** transform is called if the operation that is specified in the database transform is successful.

Use the **Return** transform in your message map to specify a nested mapping that is called if a database transform was completed successfully.

The **Return** transform provides an in-built input that provides an integer value of the number of rows that are modified by the related database transform:

Database transform	Name of input provided by Return transform	Description
Insert	NumberOfRowsInserted	The Return transform is called each time that an insert operation is successful, and reports the number of rows inserted as 1.
Update	NumberOfRowsUpdated	Number of rows updated depends on the WHERE clause
Delete	NumberOfRowsDeleted	Number of rows deleted depends on the WHERE clause

For more information, about database transforms, see “Mapping database content” on page 1523.

Select

Use the **Select** transform to retrieve data from rows in a database table, so that the data can be used in a message map.

For more information, see “Selecting data from a table” on page 1523.

When you design your message map in the IBM App Connect Enterprise Toolkit, the database server that is used during the creation of your **Select** transform is specified in the database definition file (`.dbm` file) that you selected from the Data Design project used to create the message map.

When your message map has been deployed to an integration server, and your message map is run, the database server that is used by the IBM App Connect Enterprise runtime component to process your **Select** transform is specified by the JDBC Providers policy. The JDBC Providers policy name must be the same name as the database name that is specified in your **Select** transform during development.

Submap
A **submap** enables you to use the same piece of mapping function in multiple graphical data maps.

Overview

A **submap** references another map. It calls or invokes a map from the same file or another map file, which can be stored in a library, an application, an integration service, or an Integration project.

When using submaps, consider the following points:

- A **submap** can provide callable mapping between global elements or global types from a message model.
- A **submap** cannot be used for local anonymous complex types. These must be mapped within the main map, for example, by a local map.
- You can place a submap in a project or in a library. Submaps must be placed in a project or library that is visible to the main map(s) that they are called from.

For more information, see "Creating a submap" on page 1386.

Inputs

You can connect one or more elements to the **Submap** transform with a primary connection or with a supplement connection.

- You can only use elements that are connected with a supplement connection to define conditional expressions.
- All primary inputs must have a global type.
- There is no restriction on the type of the elements that are connected with a supplement connection.

If a **Submap** has a single input that is repeating, the Graphical Data Mapping editor processes each instance that is included according to the **Cardinality** property.

If one and only one of the inputs to a **Submap** transform is repeatable, then all instances of the repeating input are processed. If you filter the indexes of the repeating input by configuring the **Cardinality** property, then only the instances that you configure are processed.

If there are multiple repeating inputs to a **Submap** transform, then you cannot use a **Submap** transform.

Define when the transform is applied at run time

You can define multiple connections between input elements and the **submap** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

You can use the inputs connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

Substring
In the Graphical Data Mapping editor, you can use the **Substring** transform to set the value of an output element to a substring of the original input value. You use this transform to extract a subset of characters that are separated by a delimiter based on a position that you indicate with an index.

Overview

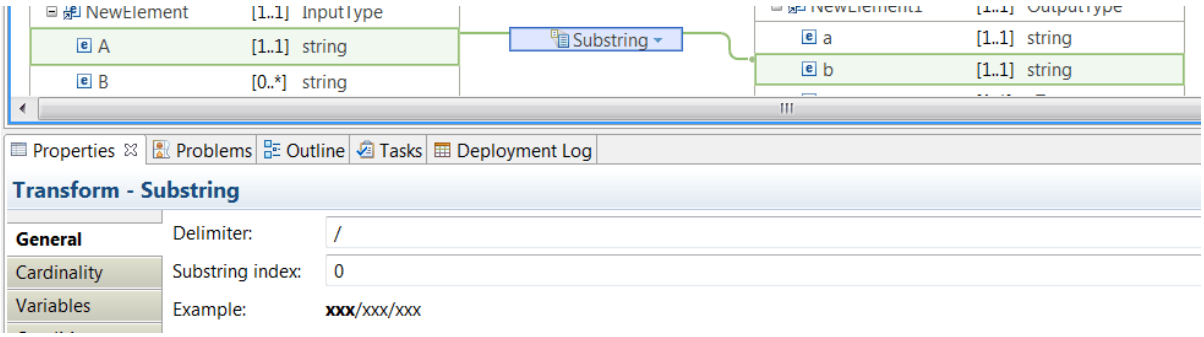
The **Substring** transform uses a delimiter and 0-based index to determine what text to extract from the incoming source string.

Based on the specified delimiter, the source string is divided into sections. The index is used to identify which section of the divided string you want to use.

For example, you can pass the following input string: `'123/124/125/126'`. You can configure the **Delimiter** property with the value `'/'`.

- If you set the **Substring index** property to `0`, the transform returns the value `'123'`.
- If you set the **Substring index** property to `1`, the transform returns the value `'124'`.
- If you set the **Substring index** property to `3`, the transform returns the value `'126'`.

By default, the **Substring index** has a value of 0, that indicates that the first section will be used.

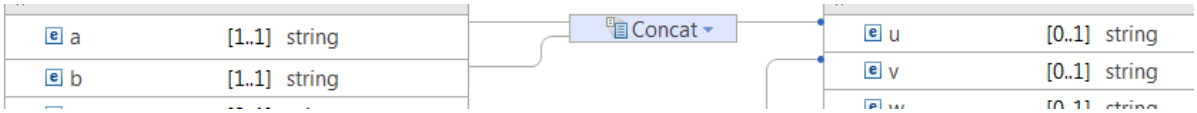


Inputs to the Substring transform

The **Substring** transform takes multiple input elements.

You wire one simple type input to the **Substring** transform as primary connection. This input contains the input value from which you need to extract part of it.

You can wire more inputs as supplementary connections. These other inputs are used to define the conditional expression that determines if a transform is applied.



Define when the transform is applied at run time

You can define multiple connections between input elements and the **Substring** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

For more information, see “Configuring the properties of a transform” on page 1431, “Defining an XPath conditional expression for a transform” on page 1433 and “Defining a Java conditional expression for a transform” on page 1439.

You can use the inputs connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Task
You can use the **Task** transform to indicate a manual task or point of concern that might need to be resolved before a message map can be used in your solution.

The **Task** transform can be connected to one or more inputs, and to a single output, to indicate the input and output elements that are required to resolve the point of concern.

Use the **General** property of a Task transform to specify the Task type:

Task type	Description
Error	Used to indicate a problem in your message map that must be resolved before the map is used in a solution. A **Task** transform of type **Error** is displayed in the Graphical Data Mapping editor, and in the **Problems** view of the Integration Development perspective. Your message map cannot be deployed if it contains one or more **Task** transforms of type **Error**.
Warning	Used to indicate an area of your message map that needs to be reviewed. A **Task** transform of type **Warning** is displayed in the Graphical Data Mapping editor, and in the **Problems** view of the Integration Development perspective. **Task** transforms of type **Warning** are ignored at run time.
Info	Used to contain a note or comment in your message map. A **Task** transform of type **Info** is displayed in the Graphical Data Mapping editor, and in the **Problems** view of the Integration Development perspective. **Task** transforms of type **Info** are ignored at run time.
To Do	Used to indicate an area of your message map that is incomplete. A **Task** transform of type **To Do** is displayed in the Graphical Data Mapping editor, and in the **Tasks** view of the Integration Development perspective. **Task** transforms of type **To Do** are ignored at run time.

Use the **Documentation** property of a **Task** transform to describe the details of your manual task or point of concern. The **Documentation** property can also contain a link to a related resource, for example a `.msgmap` file.

Using a Task transform when you convert a legacy message map

When you use the IBM App Connect Enterprise Toolkit to convert a legacy message map (`.msgmap` file) to a message map (`.map`) file, a **Task** transform is created in your new message map for each mapping structure that was not recreated by the conversion process. The **Documentation** property of a **Task** transform is used to describe the unconverted transform and, where possible, a suggested solution. For example:

```
This map transform was generated by conversion of the message mapping built in a previous
 version of the product.
The original transform from the converted file "example.msgmap_backup" was was
 "esql:concat('NativeError: ', esql:sqlnativeerror())".
The convert processing could not provide an equivalent transform. Reason: There is no supported
 corresponding XPath expression for esql:sqlnativeerror().
This map cannot be deployed and run until this 'Task' transform is re-implemented.
To re-implement this transform and remove the error marker on the map consider changing the
 'Task' transform into a <Custom XPath Transform>.
```

Update
The **Update** transform modifies one or more rows of data in a database table.

Use the **Update** transform to modify one or more rows of data that match a configured **Where** clause from a database table. Optionally, if no rows of data match your configured **Where** clause, the **Update** transform can insert a row of data. For more information, see “Updating data in a table” on page 1527.

The **Update** transform performs a single update SQL operation on the configured database server, so the inputs that you connect to your **Update** transform must provide a single set of data values. If you connect a repeating element to your **Update** transform, the Graphical Data Mapping editor moves the **Update** transform into a nested **For Each** transform.

The database server that is used during the creation of your **Update** transform is specified in a database definition file (`.dbm` file) that you select from the Data Design project used to create the map.

When your message map has been deployed to an integration server, and your message map is run, the database server that is used by the IBM App Connect Enterprise runtime component to process your **Update** transform is specified by a JDBC Providers policy. The JDBC Providers policy must have the same name as the database that is specified in your **Update** transform during development.

Built-in XPath transforms
In the Graphical Data Mapping editor, you can use built-in XPath functions to transform data.

Overview

The Graphical Data Mapping editor supports XPath functions that allow you to manipulate graphically string values, numeric values, date and time comparison, and more.

The XPath functions are grouped in the following categories:

- String functions
- Boolean functions
- Math functions
- Date and time functions
- QName functions
- Node functions
- List functions
- Diagnostic functions

For more information about XPath, see XPath tutorial or W3C XML Path Language (XPath) 2.0.

All XPath 2.0 functions are supported in the form `fn:<function_name>`.

For more information on the following XPath functions, see the following topics:

- “fn:concat” on page 1345
- “fn:string-join” on page 1348
- “fn:substring” on page 1352
- “fn:count” on page 1353
- “fn:sum” on page 1356

When you need to define complex XPath expressions, use the **Custom XPath** transform. For more information, see “Custom XPath” on page 1315.

Inputs versus arguments

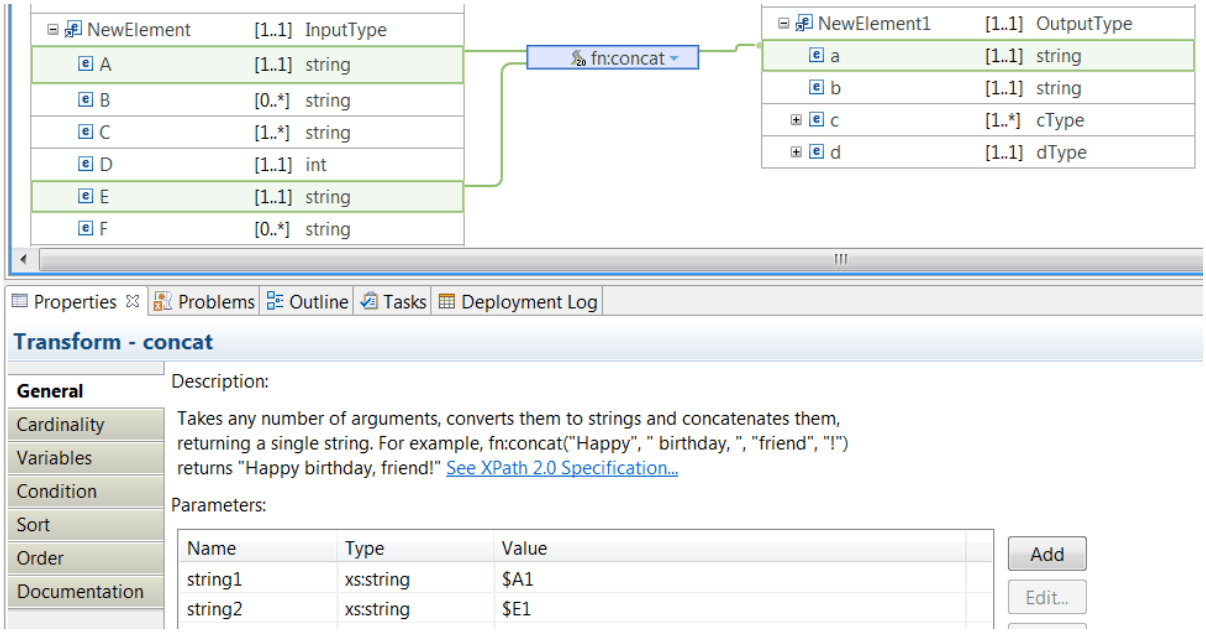
When you use an XPath transform, you must differentiate between inputs to the transform and arguments required to run the XPath function represented by the XPath transform.

Arguments are the data elements required in the calculation of an XPath function.

An argument can be a literal expression, a constant, an input element, a custom XPath expression, or a combination of multiple input elements.

You can have any number of inputs to an XPath transform. You use these inputs to define the arguments of the XPath function.

In the following figure, the XPath transform has two inputs. Each input is used as an argument of the **fn:concat** transform:

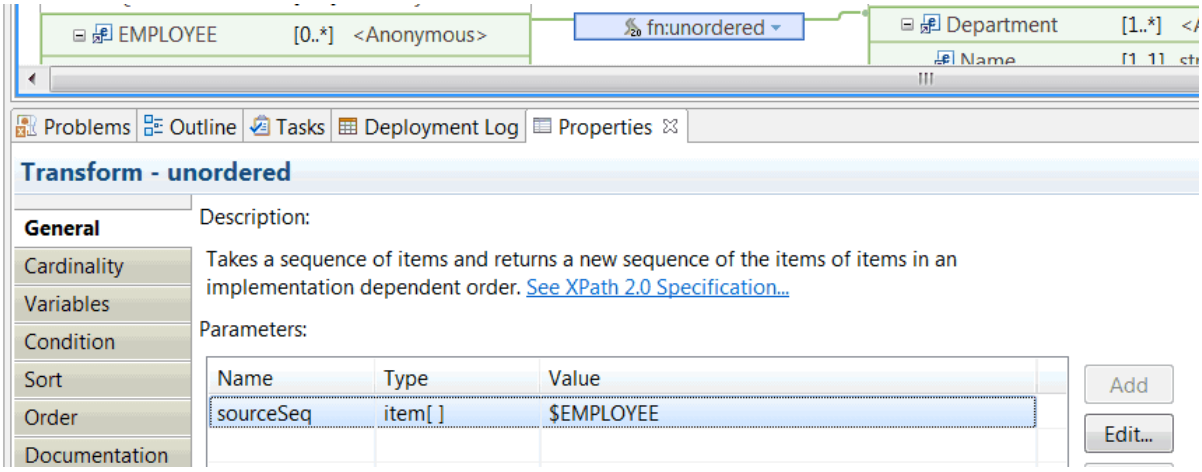


Modify the value of an input element before you apply the XPath function in a map

You can modify the value of an input element to an XPath function in the **Properties** page of the XPath transform.

You can define an XPath expression or a call to a static method on an imported Java class to modify its value. You can also create a complex expression comprising XPath, and Java.

In the **General** tab, you get the list of input elements to the XPath transform. For each element, the name, type and value are displayed.



To change the input value, you must select the element, and click **Edit**. Then you can enter the expression to modify the value.

Define when the transform is applied at run time

You can define multiple connections between input elements and an XPath transform. You can then use these input elements in a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

Alternatively, to define the conditional expression, you can call a static method on an imported Java class. You can also create a complex expression comprising XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

You configure the expression in the **Condition** tab that is available in the **Properties** page of the transform.

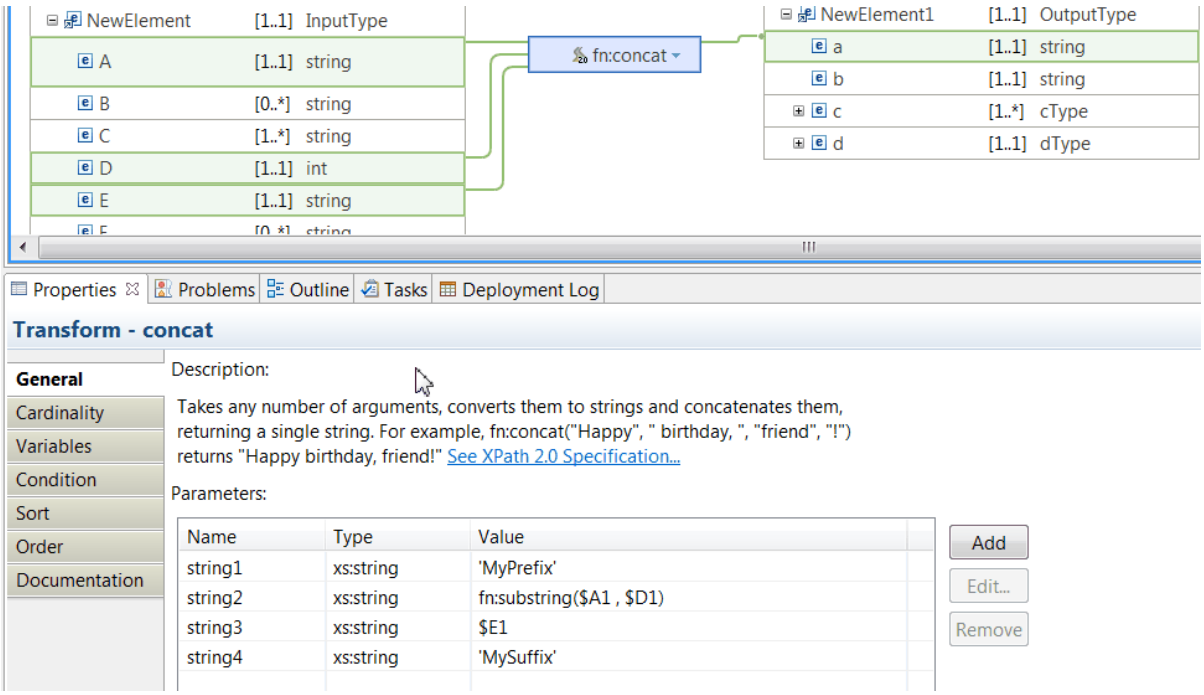
For more information, see "Configuring the properties of a transform" on page 1431, "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.

You can use the inputs connected with a primary connection to the transform. To add more inputs, you can define **supplement** connections between input elements and the transform.

Example

This example shows how to use the **fn:concat** transform to concatenate multiple input elements and set the value of a string element by using the **fn:concat** function.

The arguments to the XPath function include a prefix, two elements, and one suffix. One of the arguments is defined by an XPath function that requires data from two inputs. One of the arguments is set with data from an input. The prefix and the suffix are literals.



When you run the following message and transform the data with the **fn:concat** transform:

```
<NewElement>
  <A>My FieldA</A>
  <B>B1</B>
  <C>Field_1</C>
  <D>4</D>
  <E>FIELD_E</E>
</NewElement>
```

You obtain the following result:

```
<NewElement1>
    <a>MyPrefixFieldAFIELD_EMySuffix</a>
</NewElement1>
```

Troubleshooting

BIP3946E:

`BIP3946E: The map script generation for QName {1} has failed, with the following details: {2}`

You get **BIP3946E** when you try to deploy a map that contains an invalid XPath expression. Check the description provided by {2} to find out which expression is not a valid.

fn:concat
In the Graphical Data Mapping editor, you can use the **fn:concat** transform to create a string output element that is the result of concatenating simple input elements.

Overview

The XPath 2.0 function `fn:concat(arg1, arg2, ,,,)` takes two or more arguments, converts them to their string representation, and concatenates them, returning a single string.

The **fn:concat** transform is the representation of the **fn:concat** XPath function in the Graphical Data Mapping editor.

You can have any number of inputs to the **fn:concat** transform. These inputs are used to define the arguments of the **fn:concat** function.

For more information about XPath, see XPath tutorial or W3C XML Path Language (XPath) 2.0.

When to use the fn:concat transform

You can use **fn:concat** to concatenate input elements.

You can configure the arguments to the **fn:concat** transform to represent a prefix, a suffix, and delimiters.

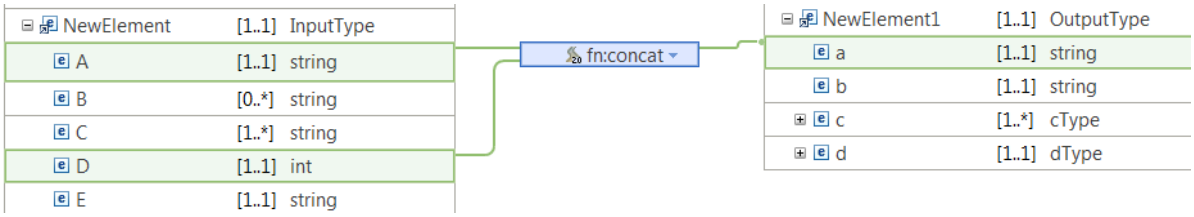
- You can use the first argument as the prefix.
- You can use the last argument as the suffix.
- You can insert a delimiter as an argument between any two arguments you want to delimit.

Inputs to the transform

You can connect simple elements to the **fn:concat** transform.

Simple input elements that are not of type **xs:string** will be cast to **xs:string**.

The following figure shows two inputs connected to the **fn:concat** transform:



Note: The Graphical Data Mapping editor does not display an error or a warning on a **fn:concat** transform when the input to the transform is not a single type element, or is repeatable.

- If the input XML has no more than one instance of the repeatable input, the **fn:concat** transform produces the expected result at run time.
- If the input XML has more than one instance of the repeatable input, the **fn:concat** transform results in a run time exception.

Arguments of the XPath function

The **fn:concat** function has two or more non-repeating arguments.

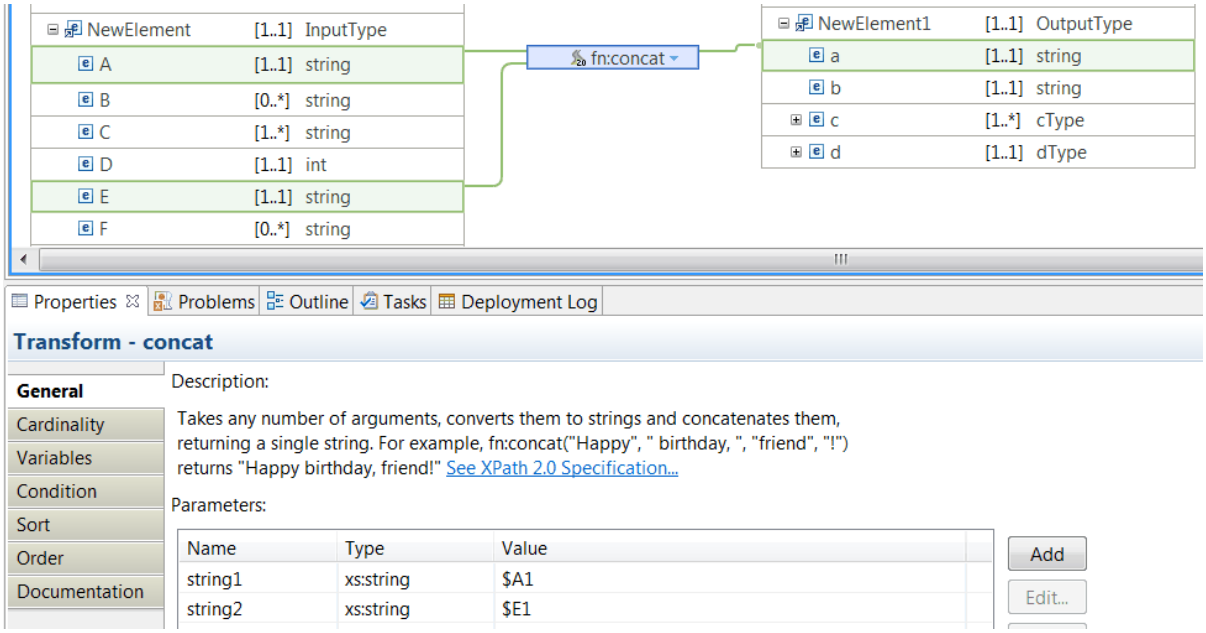
Note: A run time failure occurs complaining that a sequence is not valid if you do not ensure that each argument is non-repeating.

You define the number and the order of the arguments to the **fn:concat** function in the **General** tab of the **fn:concat** transform properties view.

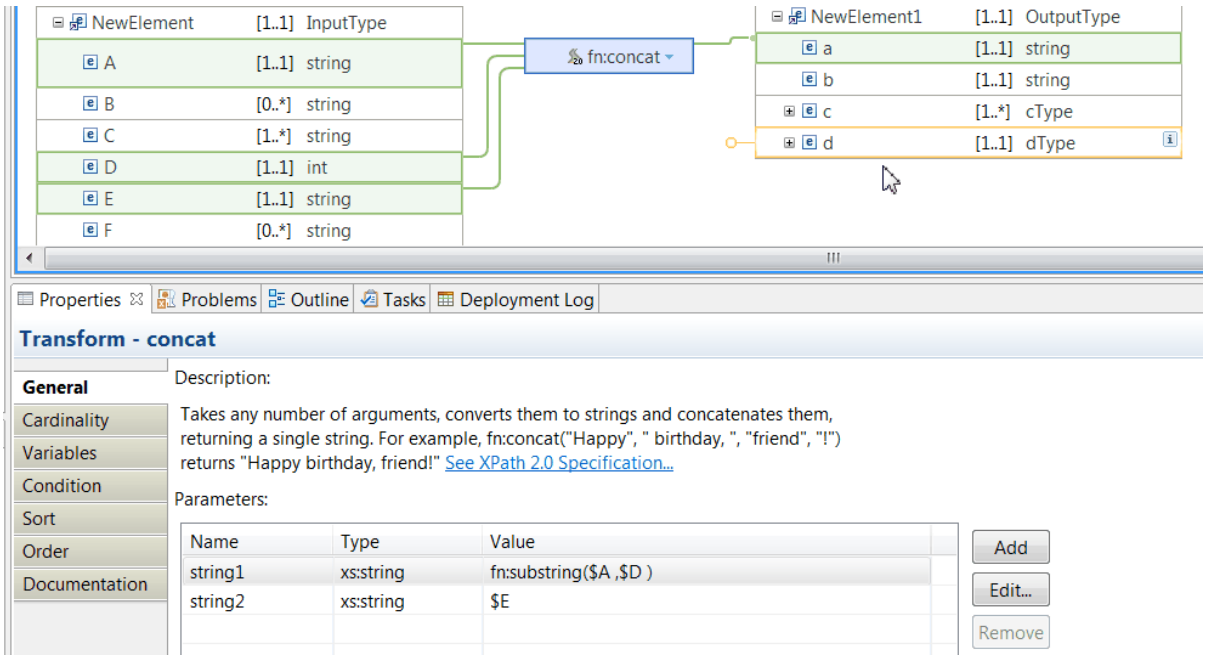
You can use the **Add** button on **General** tab to create additional arguments:

- You can add a literal value entered in single quotes.
- You can add a Custom XPath expression.

In the following figure, the **fn:concat** transform has two inputs. Each input is used as an argument of the **fn:concat** function:



In the following figure, the **fn:concat** transform has three inputs, whilst the **fn:concat** function only requires two arguments. Two of the inputs are used to define one of the arguments of the **fn:concat** transform:



Define a conditional expression

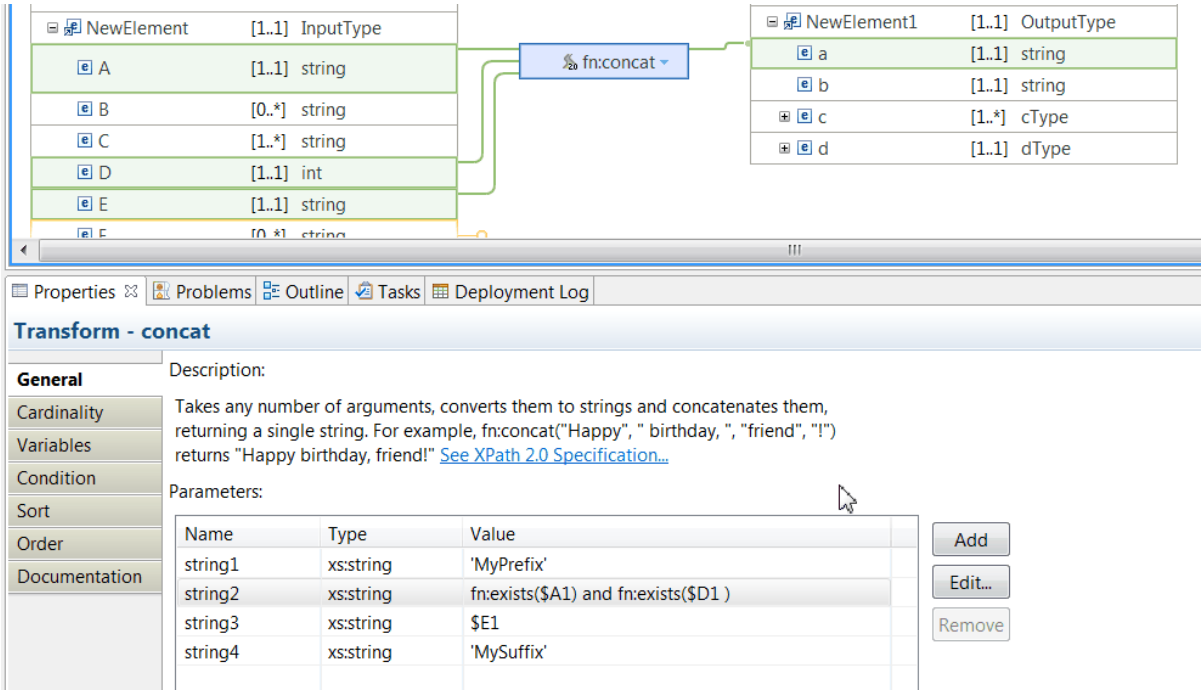
You can use any of the input elements to the **fn:concat** transform to define a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

For more information, see “Configuring the properties of a transform” on page 1431.

Example

This example shows how to use the **fn:concat** transform to concatenate multiple input elements and set the value of a string element by using the **fn:concat** function.

The arguments to the XPath function include a prefix, two elements, and one suffix. One of the arguments is set by an XPath function base on calculations from data from two inputs. One of the arguments is set with data from an input. The prefix and the suffix are literals.



When you run the following message and transform the data with the **fn:concat** transform:

```
<NewElement>
  <A>My FieldA</A>
  <B>B1</B>
  <C>Field_1</C>
  <D>4</D>
  <E>FIELD_E</E>
</NewElement>
```

You obtain the following result:

```
<NewElement1>
    <a>MyPrefixtrueFIELD_EMySuffix</a>
</NewElement1>
```

fn:string-join
In the Graphical Data Mapping editor, you can use the **fn:string-join** transform to create an output element that is the result of concatenating a sequence of input elements with a delimiter as optional.

Overview

The XPath 2.0 function `fn:string-join(arg1, arg2)` takes two arguments, converts them to their string representation, and concatenates them, returning a single string.

The **fn:string-join** transform is the representation of the **fn:string-join** XPath function in the Graphical Data Mapping editor.

You can have any number of inputs to the **fn:string-join** transform. These inputs can be used to define the arguments of the **fn:string-join** function.

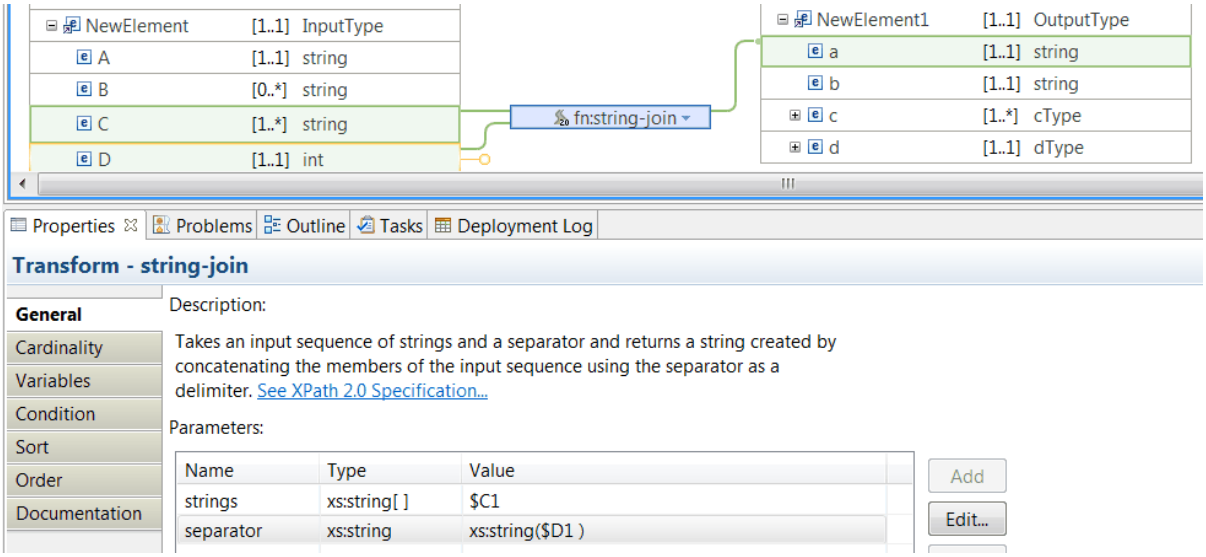
Inputs to the transform

You can connect one or more inputs to the **fn:string-join** transform. These inputs are used to define the arguments of the **fn:string-join** function.

- One of the primary inputs must be a repeatable simple element. This input is used to define the first argument of the **fn:string-join** function. It defines the sequence of elements that you want to concatenate.

  Note: When the type of the repeating elements is different from **xs:string**, the Graphical Data Mapping editor will cast the element to a **xs:string** type.
- One of the primary inputs can be a simple type element. This input is used to define the second argument of the **fn:string-join** function. It defines the delimiter used between the elements that you want to concatenate.

  Note: If the simple type is not of type **xs:string**, the transform will fail at run time with **BIP3945E**. Cast the element to a string if you want to use a non-string input element as your delimiter. For more information, see "Cast type (xs:type)" on page 1299.



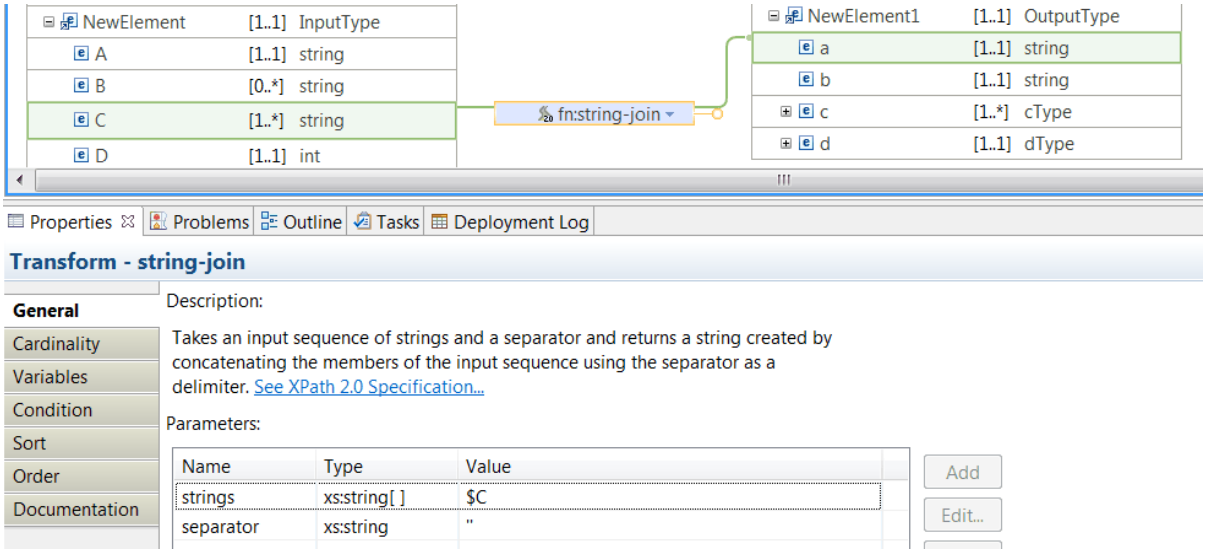
- More inputs can be connected to the **fn:string-join** transform. You can use these inputs as additional information to the transform. For example, you can use these inputs as arguments of the XPath expression that you can define in the **Condition** tab to determine whether the transform is applied at run time.

Arguments of the XPath function

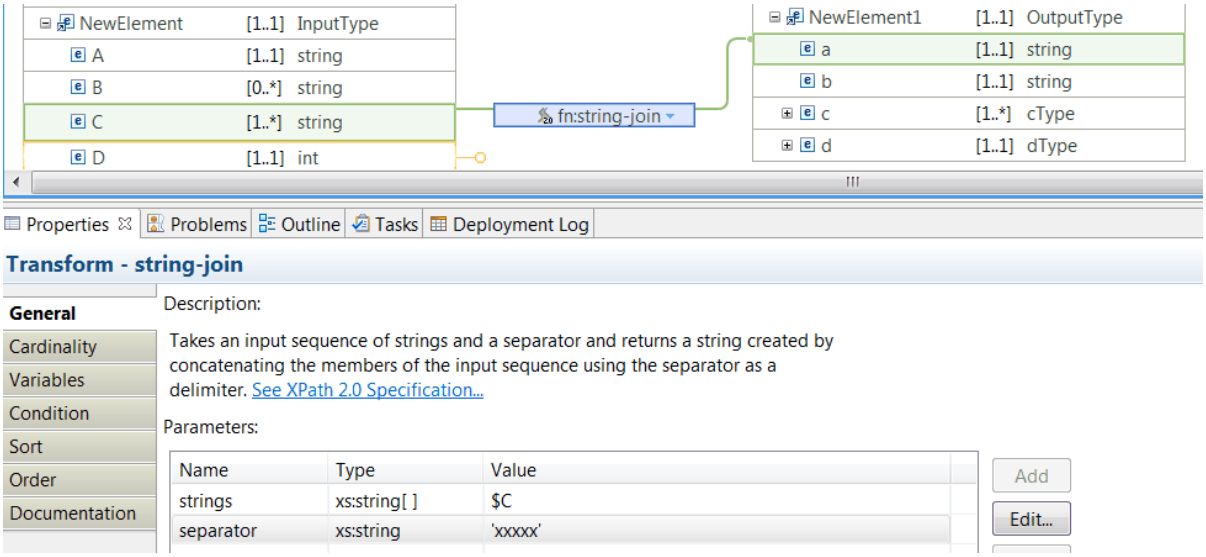
You define the arguments to the **fn:string-join** function in the **General** tab of the **fn:string-join** transform properties view.

- The first argument is a sequence of elements.
- The second argument is a delimiter. The delimiter is optional. If the separator defined for the **fn:string-join** function is a zero-length string, no delimiter is used.

In the following figure, the **fn:string-join** transform has one input. The input is used to define the first argument of the **fn:string-join** function. No delimiter is configured.



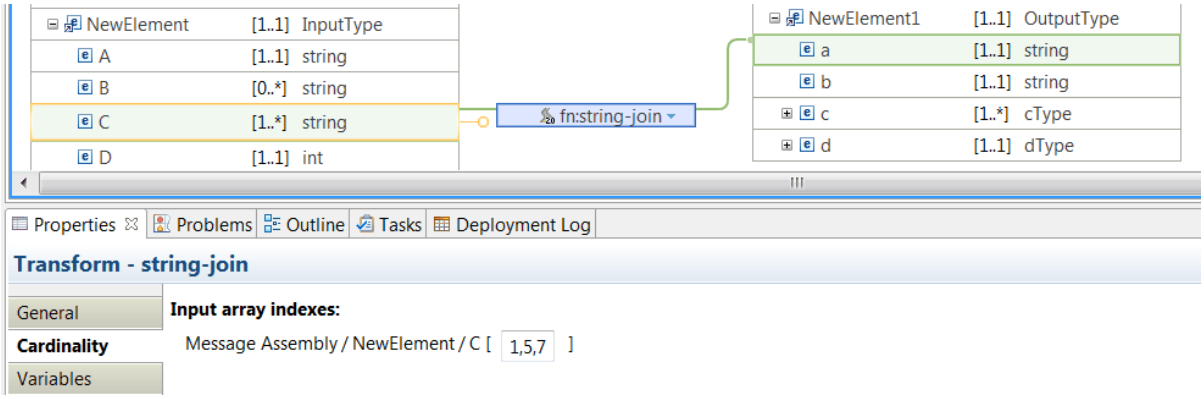
The following figure shows the **fn:string-join** transform with one repeating element as input, and a separator. To add the delimiter, you must select the **General** tab in the Properties view, click **Edit**, and then enter the value.



Cardinality

The **Cardinality** property determines the elements (also known as indexes) in the repeating input element that are processed by the **fn:string-join** transform.

You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see “Selecting the indexes of input array elements” on page 1363.



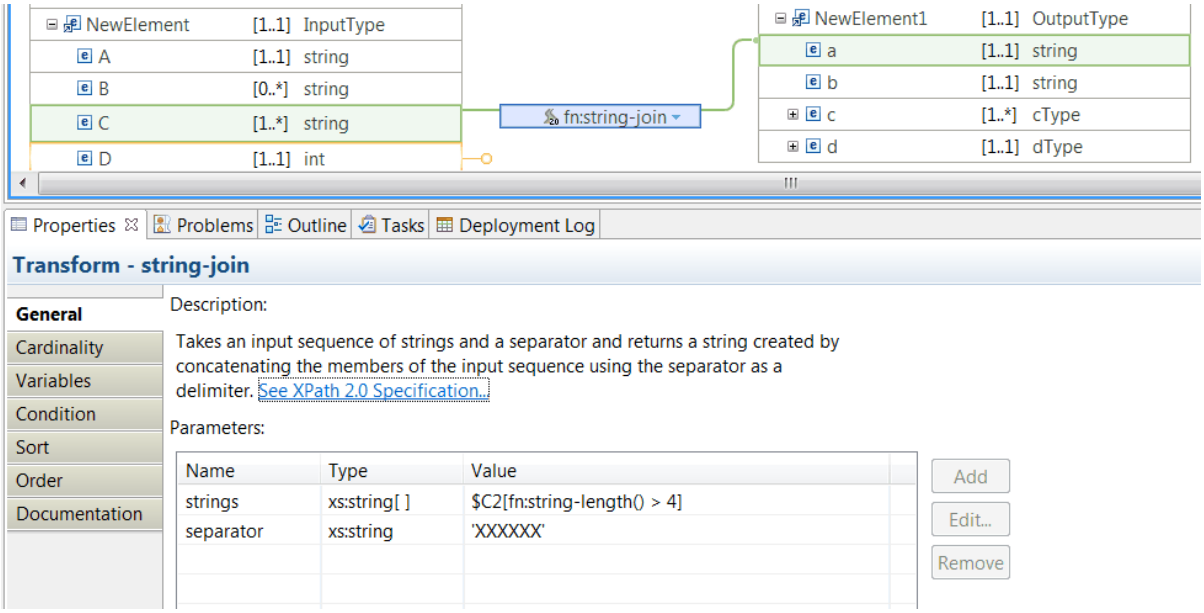
Control the instances to be concatenated

When the instances that you need to concatenate need to be calculated at run time based on a condition, you can filter dynamically the input indexes by defining a custom XPath conditional expression for the first argument named **strings** in the **General** tab. The expression determines which elements of the repeating structure are applied at run time.

For example, to calculate the value of the output element **a**, you must concatenate the elements in the repeating structure **C** whose length is greater than 4. You can use the following XPath expression:

```
$C2[fn:string-length() > 4]
```

Note: Always use content-assist to build your XPath expressions. You must use the element names used by the Graphical Data Mapping editor.



Define when the transform is applied at run time

You can use any of the input elements to the **fn:string-join** transform to define a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

For more information, see “Configuring the properties of a transform” on page 1431.

fn:substring
In the Graphical Data Mapping editor, you can use the **fn:substring** transform to set the value of an output element to a substring of the original input value. You must define the start position, and optionally, the number of characters that you need to extract.

Overview

The **fn:substring** XPath 2.0 function takes two arguments, an input string and a 1-based number, to return a part of the original string, beginning from the position indicated. You can also specify a third optional parameter as a number, to indicate the end position to compose the resulting string.

The following function call `fn:substring("12345", 2, 3)` returns `"234"`.

The following function call `fn:substring("12345", 2)` returns `"2345"`.

The `fn:substring-before()` and `fn:substring-after()` functions are variations of the `fn:substring()` function.

- Use the `fn:substring-before(arg1, arg2)` function when you need the part of arg1 that occurs before arg2 occurs in it. For example, `substring-before('1234567/CustomerID','/')` returns `1234567`.
- Use the `fn:substring-after(arg1, arg2)` function when you need the part of arg1 that occurs after arg2 occurs in it. For example, `substring-after('1234567/CustomerID','/')` returns `CustomerID`.

For more information about XPath, see XPath tutorial or W3C XML Path Language (XPath) 2.0.

Inputs to the transform

The **fn:substring** transform takes as input one simple type element. This element is used to define the first argument of the **fn:substring** function.

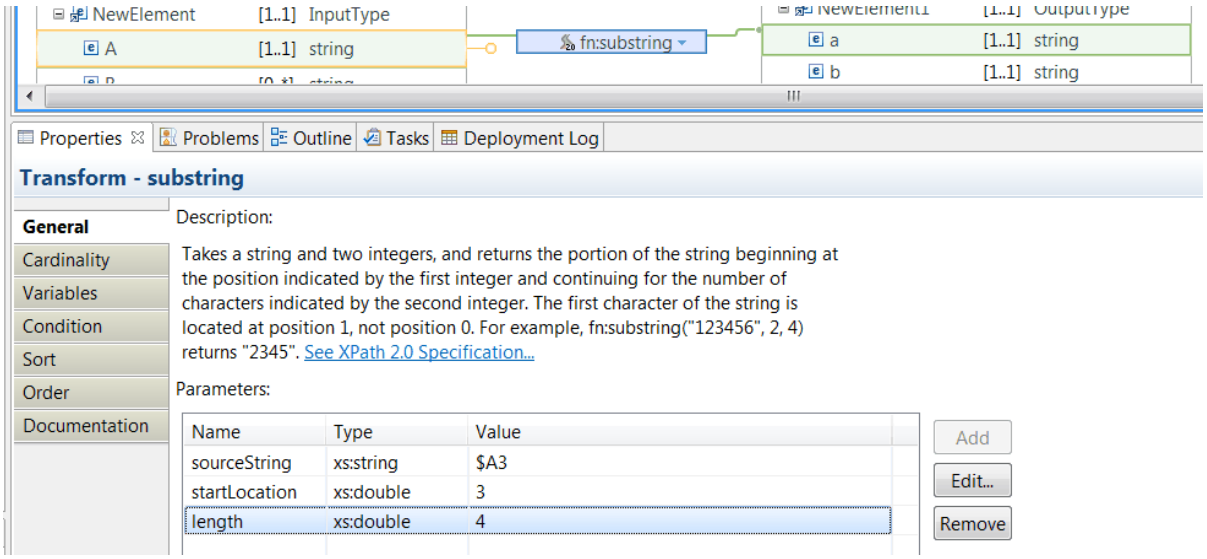
Note: If the simple type is not of type **xs:string**, the transform will fail at run time with **BIP3945E**. Cast the element to a string if you want to use a non-string input element. For more information, see “Cast type (xs:type)” on page 1299.

Arguments of the XPath function

You define the arguments to the **fn:substring** function in the **General** tab of the **fn:substring** transform properties view.

- The first argument is a string element. You can define a literal expression, a constant, an input element, or a custom XPath expression.
- The second argument is named **startLocation** and specifies the a start position.
- The third argument is named **length**, is optional, and specifies the number of characters that you need to select.

In the following figure, the **fn:substring** transform has one input. The input is used to define the first argument of the **fn:substring** function.



Define when the transform is applied at run time

You can use any of the input elements to the **fn:substring** transform to define a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

For more information, see "Configuring the properties of a transform" on page 1431.

fn:count

In the Graphical Data Mapping editor, you can use the **fn:count** transform to set the value of an output element to the total number of elements in the input element.

Overview

The XPath 2.0 function `fn:count((arg1, arg2,arg3, ...))` takes a list of elements and returns the total number of elements.

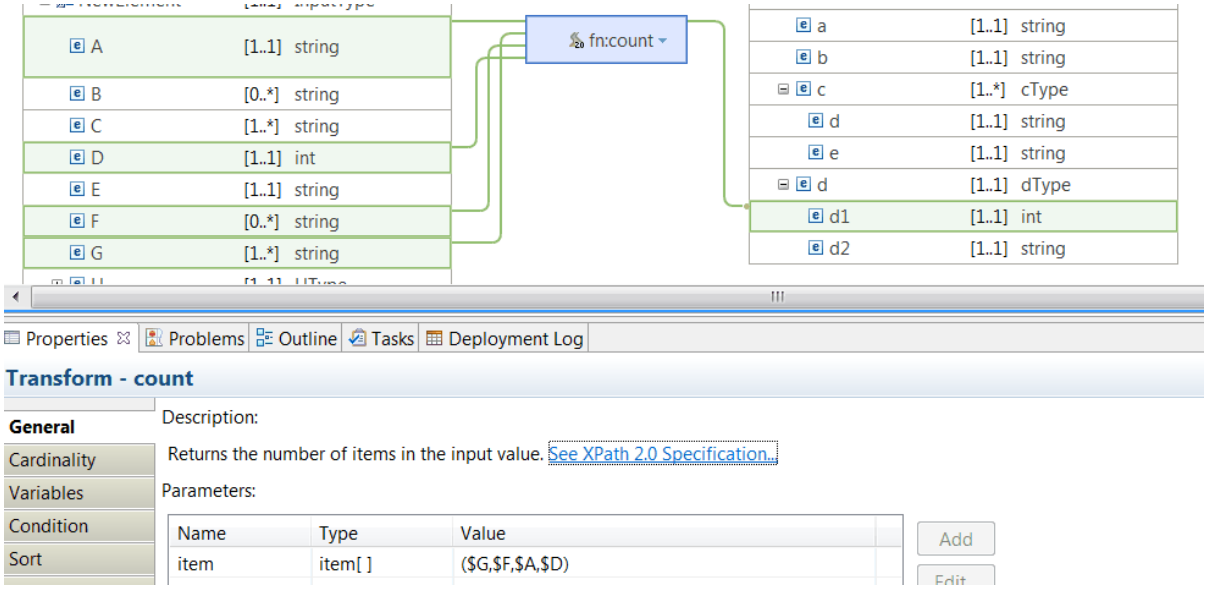
The **fn:count** transform is the representation of the **fn:count** XPath function in the Graphical Data Mapping editor.

You can have any number of inputs to the **fn:count** transform. These inputs can be used to define the arguments of the **fn:count** function.

Inputs to the transform

You can connect one or more inputs to the **fn:count** transform. These inputs are used to define the argument of the **fn:count** function.

In the following figure, there are 4 inputs to the **fn:count** transform.



When you run the following message, the value of **d1** is **4**.

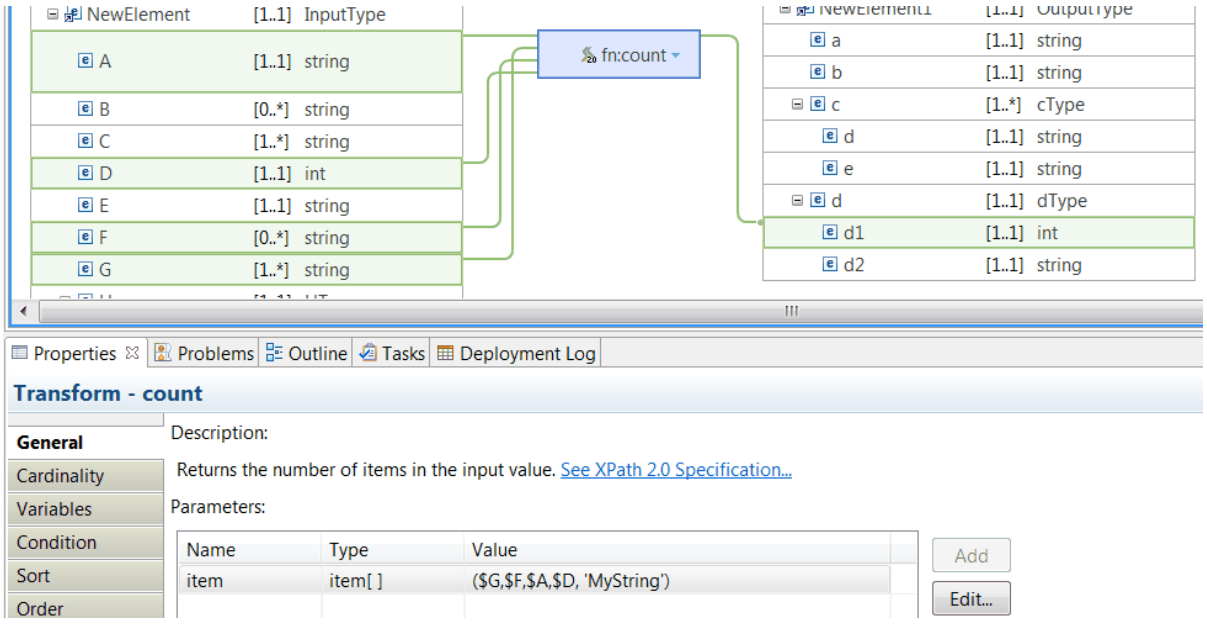
```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>10</A>
  <C>100</C>
  <C>1000</C>
  <C>10000</C>
  <D>1000</D>
  <E>112</E>
 </NewElement>
```

Arguments of the XPath function

You define the argument to the **fn:count** function in the **General** tab of the **fn:count** transform properties view.

You can define a literal expression, a constant, an input element, or a custom XPath expression as the argument.

In the following figure, there are 4 inputs to the **fn:count** transform. **A** is only considered if the string length is greater than 4. A literal expression has been added as an additional element in the argument.



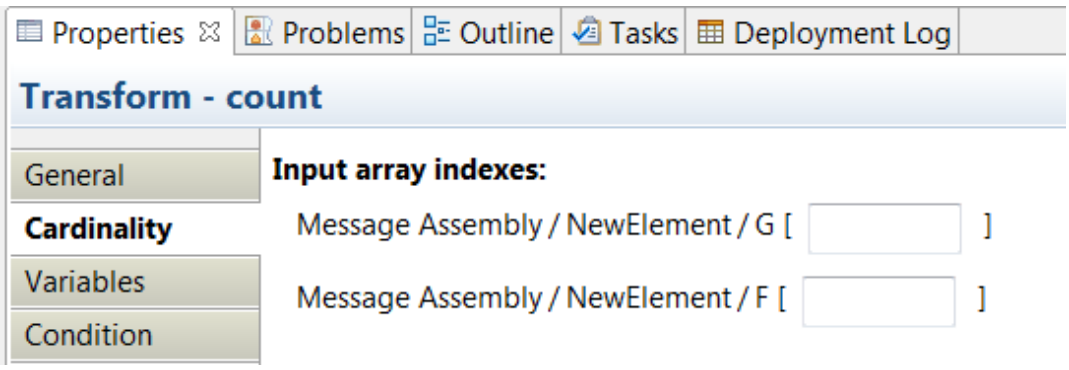
When you run the following message, the value of **d1** is **5**.

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>10</A>
  <C>100</C>
  <C>1000</C>
  <C>10000</C>
  <D>1000</D>
  <E>112</E>
</NewElement>
```

Cardinality

The **Cardinality** property determines the elements (also known as indexes) in each repeating input element that is processed by the **fn:count** transform.

You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see “Selecting the indexes of input array elements” on page 1363.



Define when the transform is applied at run time

You can use any of the input elements to the **fn:count** transform to define a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied. You define this expression in the **Condition** tab of the tranform properties.

For more information, see “Configuring the properties of a transform” on page 1431.

fn:sum

In the Graphical Data Mapping editor, you can use the **fn:sum** transform to set the value of an output element to a numeric type that is the result of the sum of all the values in a sequence. You can also use the **fn:sum** transform to set the value of an output element to the sum of durations in a sequence.

Overview

You can use the XPath 2.0 function `fn:sum(arg1, arg2)` to sum numeric or duration values in a sequence.

The **fn:sum** transform is the representation of the **fn:sum** XPath function in the Graphical Data Mapping editor.

You can have any number of inputs to the **fn:sum** transform. These inputs can be used to define the arguments of the **fn:sum** function.

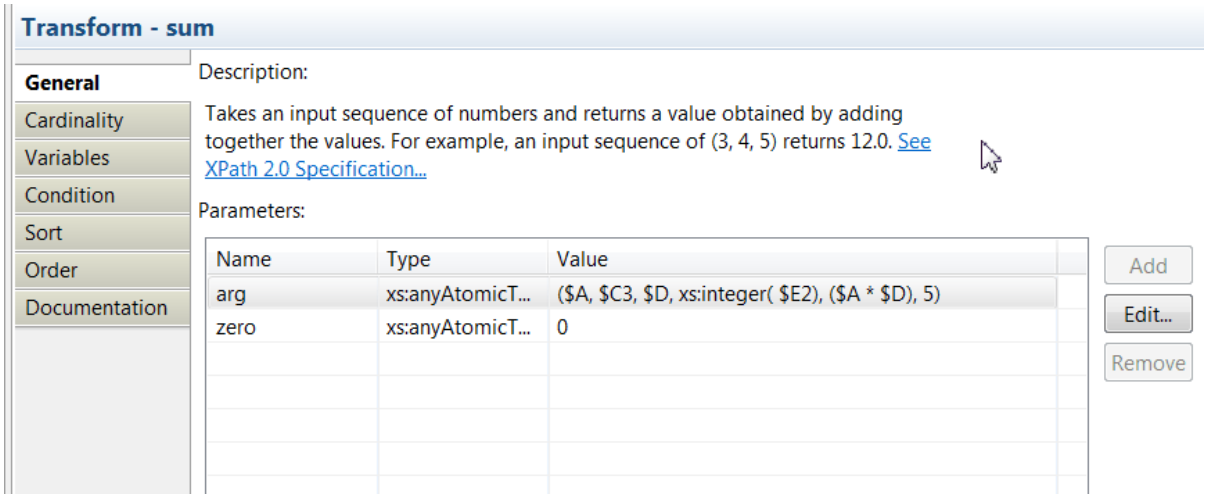
Inputs to the transform

You can connect one or more inputs to the **fn:sum** transform. These inputs are used to define the arguments of the **fn:sum** function.

Arguments of the XPath function

You define the two arguments to the **fn:sum** function in the **General** tab of the **fn:sum** transform properties view.

- The first argument (**arg**) contains any number of numeric and untyped values, or any number of duration values.
- The second argument (**zero**) contains the default value returned by **fn:sum** when the sequence is empty. You can set this value to the integer 0, a duration of zero seconds, or any other atomic value.



You must consider the following points when you define the arguments to the **fn:sum** function:

- Untyped values are cast to a double type element.
- You can edit the sequence specified for **arg** and then add literal values, cast untyped values, and add more values that are the result of arithmetic operations.
- If **arg** is an empty sequence, and **zero** is set, then **fn:sum** returns the value specified by **zero**.
- When you sum durations, you must ensure that all the elements in the sequence have the same type, that is, all values are of type **xs:yearMonthDuration** or all values are of type **xs:dayTimeDuration**.

Note: If an input is not of a numeric type such as **xs:int**, the transform will fail at run time. Cast the input element to a numeric type. For more information, see "Cast type (xs:type)" on page 1299.

Cardinality

The **Cardinality** property determines the elements (also known as indexes) in the repeating input element that are processed by the **fn:sum** transform.

You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see "Selecting the indexes of input array elements" on page 1363.

Define when the transform is applied at run time

You can use any of the input elements to the **fn:sum** transform to define a conditional expression that defines the condition under which the transform is applied. If the condition evaluates to **true**, the transform is applied.

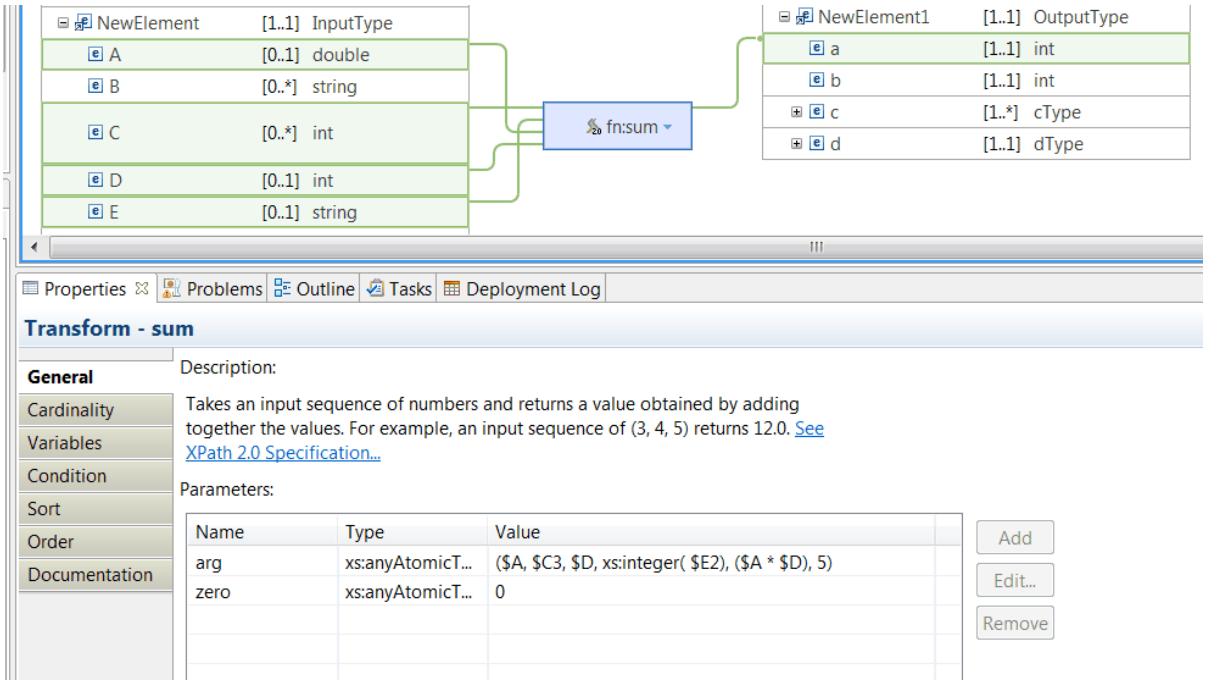
For more information, see "Configuring the properties of a transform" on page 1431.

Example: Calculate the value of a numeric output element

In the following figure, the **fn:sum** transform has four inputs and the **fn:sum** function has two arguments.

The input **E** is of type string and is cast to an integer type.

The second argument **zero** is set to the value of multiplying two input elements.



The **fn:sum** transform returns **11147** for the following input message:

```
<?xml version="1.0" encoding="UTF-8"?>
<NewElement>
  <A>10</A>
  <C>2</C>
  <C>10</C>
  <D>1000</D>
  <E>120</E>
</NewElement>
```

Choosing a transform to set the value of a simple type output element
You can use different transforms, such as the **Assign** transform, the **Create** transform, the **Move** transform, or the **xs:type** transform, to set the value of an output element.

Procedure

Complete the questionnaire to identify the transform that you can use to set the value of an output element:

1. Do you want to set the value to a fixed value?

   Use the **Assign** transform.
2. Do you want to initialize the output element, that is, do you want to create an empty structure?

   Use the **Create** transform to initialize a string output element or a hexBinary output element.
3. Do you want to create a nil output element?

   Use the **Create** transform and verify that the output element is defined as `nillable="true"` in the schema.
4. Do you want to set the output element as nil by using an input element with a value of nil?

   Use the **Move** transform. Verify that the input element is defined as `nillable="true"` in the schema. Ensure that the value of the input element is nil.
5. Do you want to set the output element to a default value?

   Use the **Create** transform and verify that the output element has a default value set in the schema.
6. Do you want to set the output element with input from a database column?

   Use the **Select** transform to obtain the database input value. Then, in the nested map that is associated with the **Select** transform, use the **Move** transform to set the value of the output element.
7. Do you want to set the value of the output element with the value of an input element? Do the input element and the output element have the same type associated? Do you want to cast the input value to the type of the output value?

   - When the input and output element have the same type, use the **Move** transform.
   - When the input and output element have different data types, use the **xs:type** transform.
8. Do you want to calculate the value of the output element by using the value of one or more input elements?

   To set an output element with a string data type or hexBinary data type, use any of the following transforms:

   - **Concat**
   - **Normalize**
   - **Append**
   - **Substring**
   - **fn:string-join**
   - **Custom XPath**
   - **Custom Java**
   - **Custom ESQL**

   To set up an output element with any other data type, use any of the following transforms:

   - Any supported XPath functions, for example, **fn:round**
   - **Custom XPath**
   - **Custom Java**
   - **Custom ESQL**

9. Do you want to apply the transform always? Do you want to apply the transform when a condition based on input data occurs?

   Define a conditional expression for the transform you choose. This expression determines when the transform is applied. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.

Results

Table 53. Setting the value of a simple output element

	Number of input elements that are required to set the value of the output element	Transforms to set a string data type, or a hexBinary data type	Transforms to set other simple data types
Set the output element with a fixed value	0	**Assign**	**Assign**
Initialize the output element	0	**Create**	not valid option
Set the output element as nil	0	**Create** Condition: The output element must be defined as `nillable="true"` in the schema.	**Create** Condition: The output element must be defined as `nillable="true"` in the schema.
Set the output element as nil by using a nil input element	1	**Move** Condition: The input element must be defined as `nillable="true"` in the schema.	**Move** Condition: The input element must be defined as `nillable="true"` in the schema.
Set the output element with a default value	0	**Create** Condition: The output element must have a default value set in the schema.	**Create** Condition: The output element must have a default value set in the schema.
Set the output value from a database table column	1..N	**Select** transform to obtain the database input value and **Move** transform to set the value	**Select** transform to obtain the database input value and **Move** transform to set the value
Copy the value of the input element to the output element (both elements have the same data type)	1	**Move**	**Move**

Table 53. Setting the value of a simple output element (continued)

	Number of input elements that are required to set the value of the output element	Transforms to set a string data type, or a hexBinary data type	Transforms to set other simple data types
Copy the value of the input element to the output element (elements have different data types)	1	**xs:type**	**xs:type**
Calculate the value by using the values of multiple input elements	1..N	**Concat**, **Normalize**, **Append**, **Substring**, **fn:string-join**, **Custom XPath**, **Custom Java**, **Custom ESQL**	XPath **fn:** functions, **Custom XPath**, **Custom Java**, **Custom ESQL**

What to do next

Learn about the transforms. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Choosing a transform to set the value of a complex output element
You can use different transforms, such as **Move**, **Create**, **Assign**, to set the value of a complex output element.

Procedure

Complete the questionnaire to identify the transform that you can use to set the value of a complex output element:

1. Do you want to set the value to a fixed value?

   Use the **Assign** transform.
2. Do you want to initialize the output element, that is, do you want to create an empty structure?

   Use the **Create** transform.
3. Do you want to create a nil output element?

   Use the **Create** transform and verify that the output element is defined as `nillable="true"` in the schema.
4. Do you want to set the output element as nil by using an input element with a value of nil?

   Use the **Move** transform. Verify that the input element is defined as `nillable="true"` in the schema. Ensure that the value of the input element is nil.
5. Do you want to set the output element to a default value?

   Use the **Create** transform and verify that the output element has a default value set in the schema.
6. Do you want to set the output element with input from a database column?

   Use the **Select** transform to obtain the database input value. Then, in the nested map that is associated with the **Select** transform, use the **Move** transform to set the values of the output element.

7. Do you want to set the value of the output element with the value of an input element? Do the input element and the output element have the same type associated? Do you want to cast the input value to the type of the output value?
   - When the input and output element have the same type, use the **Move** transform.
   - When the input and output element have different data types, use the **xs:type** transform to set the value of each element.
8. Do you want to apply the transform always? Do you want to apply the transform when a condition based on input data occurs?

   Define a conditional expression for the transform you choose. This expression determines when the transform is applied. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.

Results

Table 54. Setting the value of a complex output element

	Number of input elements that are required to set the value of the output element	**Transforms to set a complex type**
Set the output element with a fixed value	0	**Assign**
Initialize the output element	0	**Create**
Set the output element as nil	0	**Create** Condition: The output element is defined as `nillable="true"` in the schema.
Set the output element as nil by using a nil input element	1	**Move** Condition: The input element must be defined as `nillable="true"` in the schema.
Set the output element with a default value	0	**Create** Condition: The output element has a default value set in the schema.
Set the output value from a database table column	1..N	**Select** transform to obtain the database input value and **Move** transform to set the value
Copy the value of the input element to the output element (both elements have the same data type)	1	**Move**
Copy the value of the input element to the output element (elements have different data types)	1	**xs:type**

What to do next

Learn about the transforms. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Choosing a transform to map repeating elements
In the Graphical Data Mapping editor, you can use the **For Each** transform, the **Append** transform, the **Join** transform, XPath functions, the **Custom XPath**, and the **Custom Java** transform to map input and output arrays. You can use any of these transforms to choose the set of the elements in the array that you want to transform, and place the result in an output array or single element.

About this task

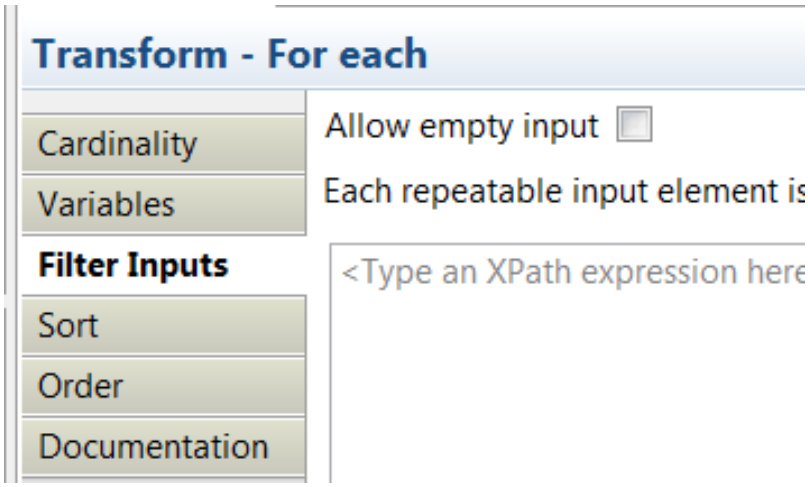
The only way to control the order of populating an output array is by using the **Append** transform. You must provide input connections to the **Append** transform and set their order in the **Properties** page. For more information, see "Append" on page 1285.

Procedure

Choose one of the following transforms to map repeating elements:

1. When you have repeating elements as input and output to a transform, you can use the **For Each** transform to set the output array element.

   The **For Each** transform iterates over one input array element, which can be either a simple type or a complex type, and enters a nested mapping in which you can provide transforms to populate an instance of the output from the input. You can configure the cardinality to filter which instances to process based on index. Also, you can provide a Boolean expression which will be applied to each instance to determine if it will be mapped.

   Additionally, you can also set the **Allow empty input** option so that the Graphical Data Mapping editor enters the nested transform once when no matches occur. This can be used to implement an outer join by providing a supplementary input to the **For Each**.

   For more information, see "For Each" on page 1320.
2. When you have multiple inputs that are either simple type elements, complex type elements, or repeating elements from which you need to construct an array, you can use the **Append** transform to add instances to either a simple type output array or a complex type output array.

   For more information, see "Append" on page 1285.
3. When you have multiple inputs that are either simple type elements, complex type elements, or repeating elements as input to a transform, and you want to join these elements, you can use the **Join** transform to combine them into a single repeating output element. The output element can be an

array or a single element. You configure an expression to control the match conditions for the join. The Graphical Data Mapping editor provides a link to create a simple match by index expression.

For more information, see "Join" on page 1332.

4. You can use XPath functions to map from an array to a single element. For example, you can use **fn:string-join** to return a string created by concatenating multiple string arguments or **fn:sum** to return the sum of a repeating numeric element into a single element.

   For more information, see "Built-in XPath transforms" on page 1342.

5. When you want to map a particular instance from an array to a single output you can use a **Custom XPath** transform that has an XPath predicate expression to select a particular instance.

   For example, if you want to select the "Country" value from the instance of an array of "Address" elements that has a child element "Type" with the value "home", use the following form for your Custom XPath transform:

   ```
   $Address[Type = 'home']/Country
   ```

   Note: If you want to use an input element value to dynamically select an element of an array by index, you must ensure that your XPath predicate expression yields a singleton numeric value. For example, to select the "Country" value from the instance of an array of "Address" elements at the index that is specified by an input element "Counter" (that is defined as `xsd:int`), use the following form for your Custom XPath transform:

   ```
   $Address[fn:data($Counter)]/Country
   ```

   The `fn:data()` function is used to obtain the numeric value of the input element. If the "Counter" element is defined as `xsd:string` you must use the following form to explicitly cast the value to a numeric value:

   ```
   $Address[xs:int($Counter)]/Country
   ```

6. You can use a **Custom Java** transform to map an array by passing the array as an input or output parameter using a list of MbElement objects.

   For more information, see "Custom Java" on page 1312.

Selecting the indexes of input array elements

When you are transforming array elements in the Graphical Data Mapping editor, you can use the **Cardinality** page of the properties view to select the indexes of the input elements that you want the transform to operate over.

When you restrict the indexes of an input array, you reduce the number of indexes processed by the transform. This helps improve performance.

To specify the indexes, enter a value in the **Input array indexes** field of the Cardinality properties page for the transform.

The following table shows examples of the values that you can enter:

Table 55. Example values of input array indexes

Selected index elements	**Value in input array indexes field**
All indexes	* (or leave empty)
Only index 5	5
indexes 1 - 3	1:3
indexes 1, 3, and 5	1,3,5
indexes 2 and up	2:*
indexes 1, 3, 5 and up	1,3,5:*

Table 55. Example values of input array indexes (continued)	
Selected index elements	**Value in input array indexes field**
indexes 2 - 8, but not 5	2:4,6:8
All indexes except 5	1:4,6:*

The indexes are 1-based, which means that the first element of the array is referenced as 1, the second element as 2, and so on. If the cardinality field is left blank for a specified array, all indexes are taken. If you have multiple levels of nested array elements, blank cardinality fields imply that all indexes are taken. Therefore, if the input element to a transform is `A/B[]/C[]`, where B and C are arrays with no indexes specified, all indexes are taken. This means that all C's part of B[1], all C's part of B[2], all C's part of B[3], and so on, are taken.

Choosing a transform to concatenate input data
In the Graphical Data Mapping editor, you can use a **Concat** transform, a **Custom XPath** transform, or an XPath function to define a mapping operation that sets the value of an output element by concatenating input data.

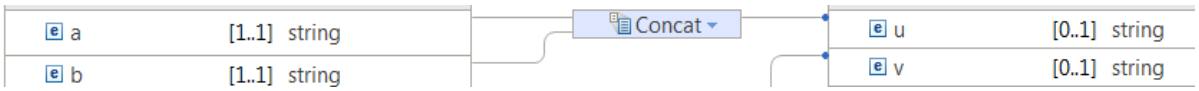
Procedure

Choose one of the following transforms to concatenate multiple input values and set the value of an output element:

- If your input elements are simple non-repeatable elements, and you want to create an output element that is the result of concatenating these input elements with a common delimiter, use the **Concat** transform. The delimiter is optional.

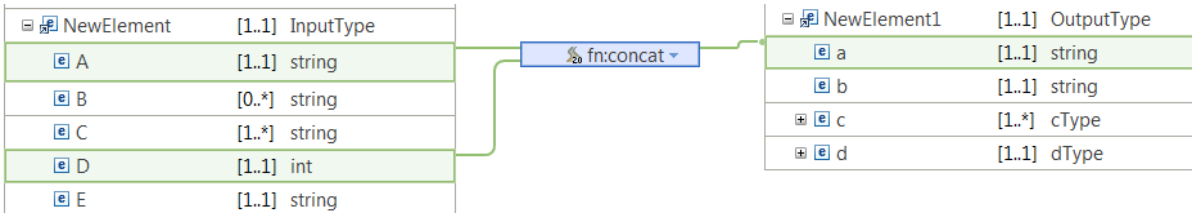
  The **Concat** transform concatenates two or more simple non-repeatable elements from the input into a single string value in the output. You can specify a constant to define a prefix, a suffix, and multiple delimiters.

  For more information, see "Concat" on page 1301.



- If your input elements are simple non-repeating elements, and you want to create an output element that is the result of concatenating these input elements by using an XPath expression with different delimiters, use the **fn:concat** transform. The delimiters are optional.

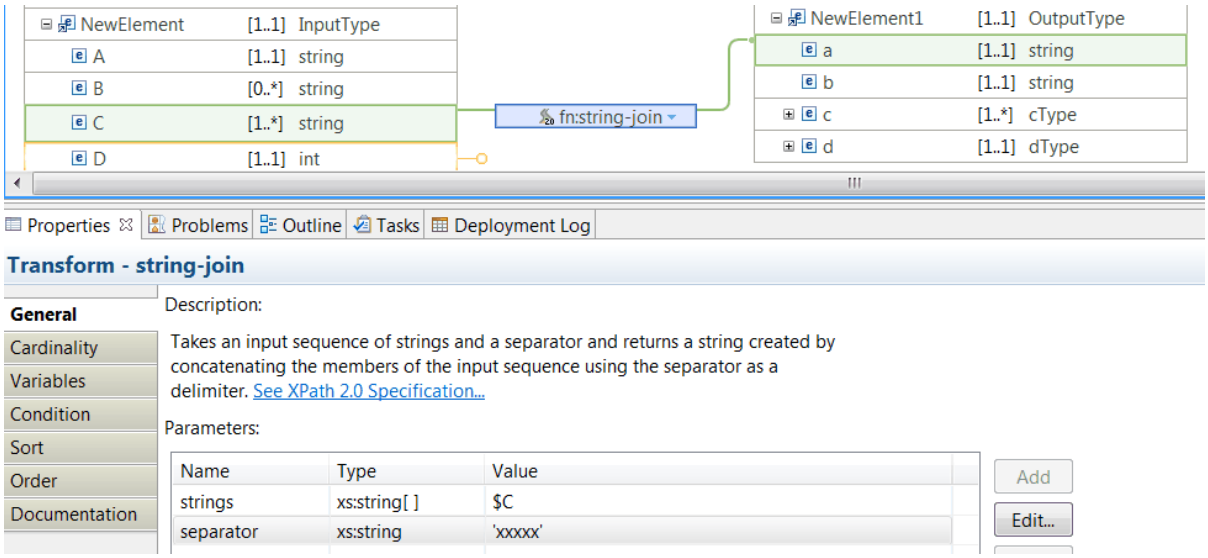
  For more information, see "fn:concat" on page 1345.



- If the input is a repeatable simple element, and you want to create an output element that is the result of concatenating the sequence of input elements with a delimiter as optional, use the **fn:string-join** transform.

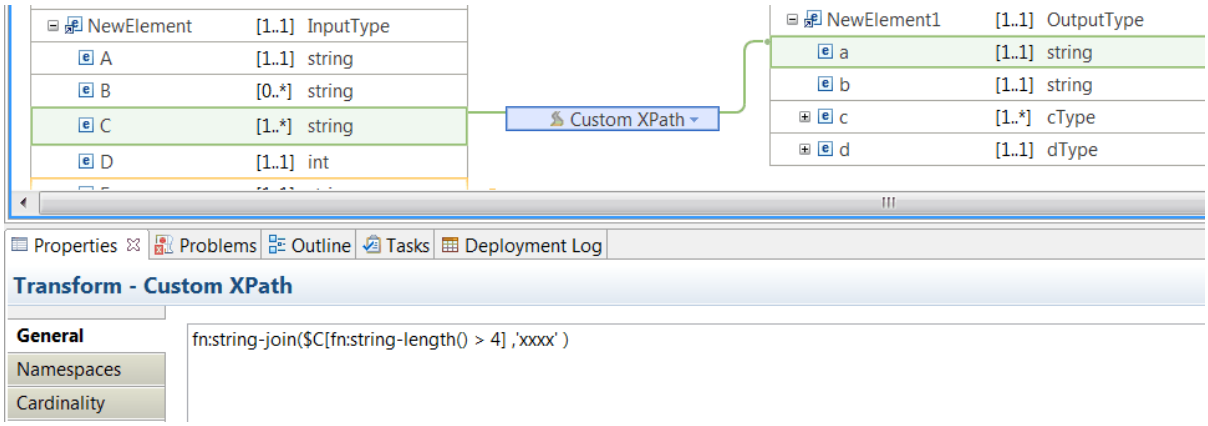
  For more information, see “fn:string-join” on page 1348.

  The following figure shows the **fn:string-join** transform:



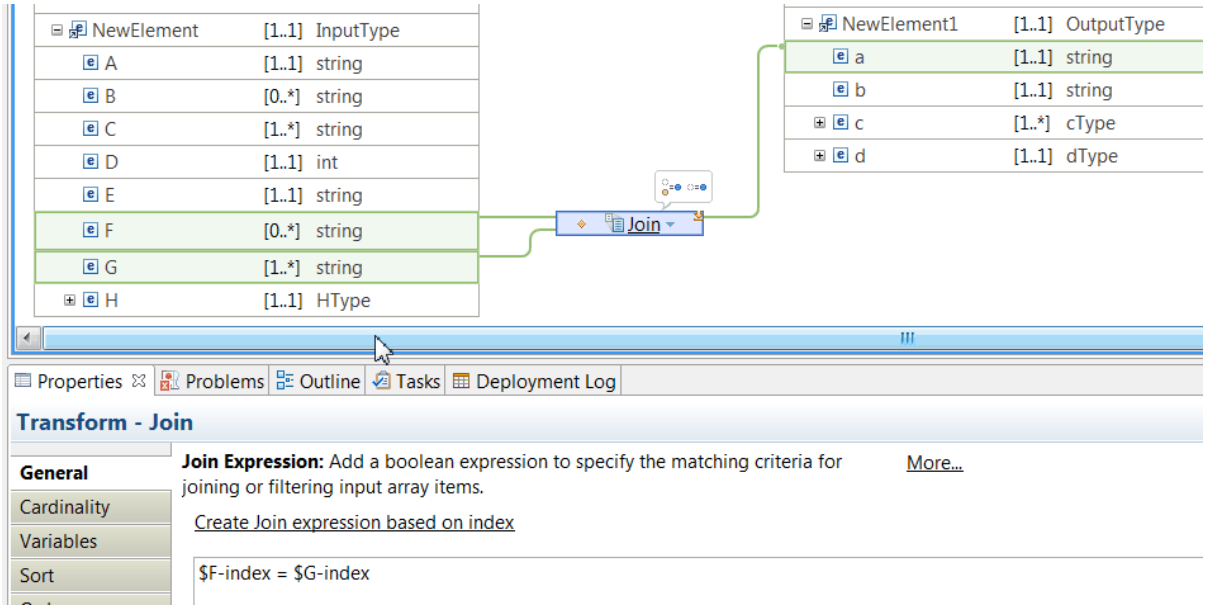
- Use the **Custom XPath** transform to define an XPath expression that concatenates multiple input elements into a simple output type.

  The following figure shows a **Custom XPath** transform that concatenates some input elements in the array into a string element:



- If you need to specify the matching criteria for joining or filtering input elements, use the **Join** transform. Then, define transforms between the input and output elements in the nested map associated to the **Join** transform.

  For more information, see "Join" on page 1332.



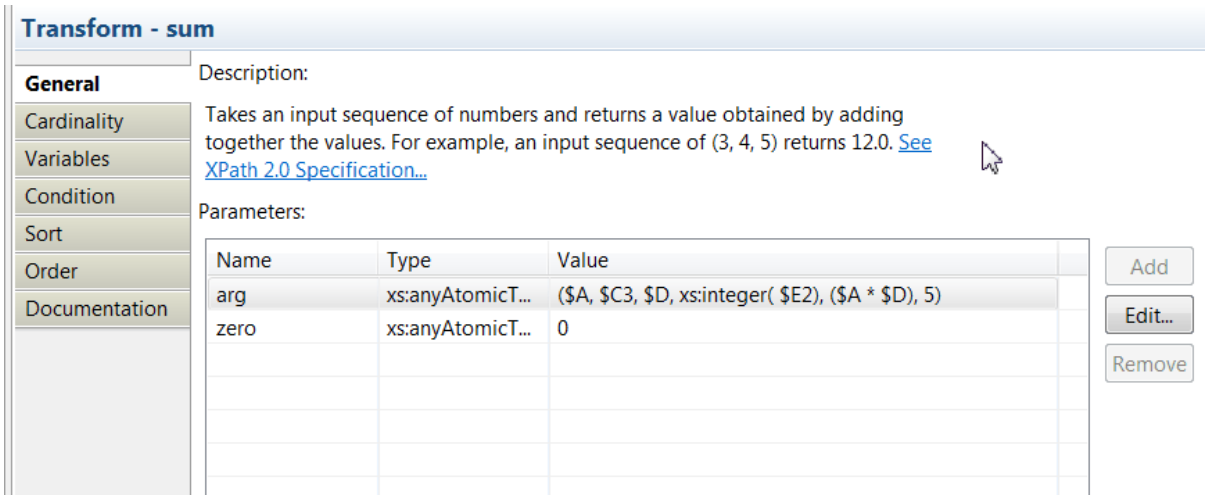
Choosing a transform to perform an arithmetic operation
In the Graphical Data Mapping editor, use the **fn:sum** or a **Custom XPath** transform to implement an arithmetic operation.

Procedure

Choose one of the following transforms to implement an arithmetic operation to set the value of an output element:

- Use the **fn:sum** transform to calculate the sum of the values of multiple input elements.

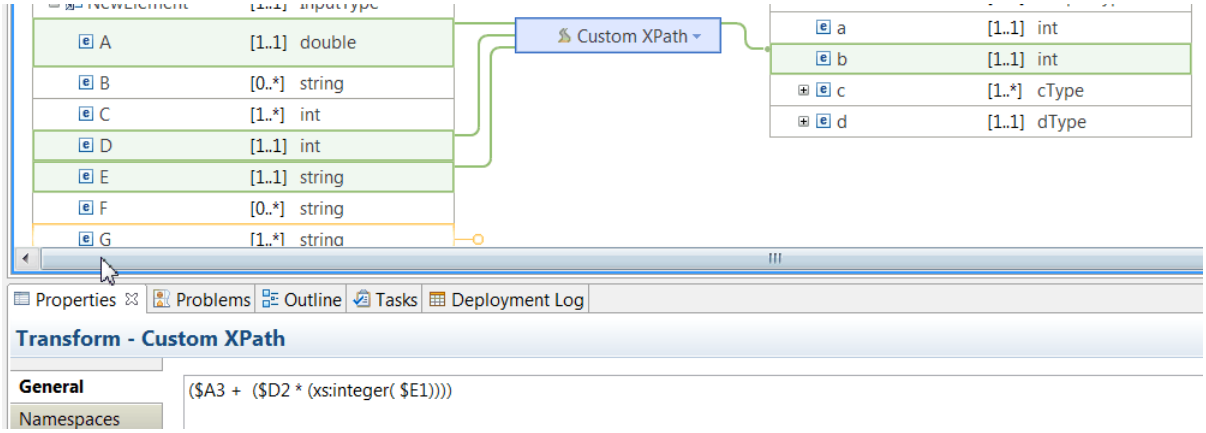
  For more information, see "fn:sum" on page 1356.



- Use the **Custom XPath** transform to implement any of the following arithmetic operations: addition (+), subtraction (-), multiplication (*), division (div), or any combination of these operations.

  Note: You can only use as arguments of an arithmetic operation non-repeating simple type input elements.

  For more information, see “Custom XPath” on page 1315.



Choosing a transform to define a conditional mapping
In the Graphical Data Mapping editor, you can define conditional transformations by using a transform or by defining a conditional expression in the transform.

Procedure

Choose any of the following options to configure a conditional mapping in your map:

1. Use the **If** transform, the **Else if** transform, and the **Else** transform to control the flow of the mapping.

   If, **Else if**, and **Else** operate as a group of conditional transforms. The condition is applied to the input element of the conditional transform. If the condition is satisfied, the nested transform within the conditional transform is run.
2. Use a **Custom Java** transform or a **Custom ESQL** transform to provide a condition function

   For more information, see “Custom Java” on page 1312 and “Custom ESQL” on page 1308.
3. Define a conditional expression in a transform to determine when the transform is applied bu configuring the **Condition** tab of the transform.

   For more information, see “Defining an XPath conditional expression for a transform” on page 1433 or “Defining a Java conditional expression for a transform” on page 1439.

   Note: If you use a condition to optionally create one or more child output elements within a complex element, check that the condition is on the transform targeting the whole complex element, not just a child. If the conditional transform only targets a child, the folder element within the complex element will be created in the output before the condition is evaluated.

What to do next

For more information about mapping transforms, see “Transform types in the Graphical Data Mapping editor” on page 1282.

Choosing a transform to map an input message to multiple output messages
You can create a map that takes a single input message and produces either multiple instances of an output message model, or one or more instances of different output message models.

About this task

When you create a map in the **New Message Map** wizard, you can only select a single input and a single output. However, you can use the **Add output** button in the Graphical Data Mapping editor to add additional outputs.

Note: You can only split an input message into multiple output messages in main maps. This is not valid in submaps.

Procedure

Choose any of the following options to split a message in your map:

- To produce multiple instances of a particular output message, you can use the **For Each** transform or the **Join** transform.

  A typical use of this function is message splitting, in which an input batch message is divided into individual record messages.

  When you run the map, a new message is propagated for each iteration of the **For Each** or **Join** transform.

  For more information, see "Splitting an input message into multiple output messages" on page 1503.
- To produce one or more instances of different output messages, you can use the **If/Else** transform.

  When you run the map, a new message is propagated for each conditional transform that is applied.

  For more information, see "Mapping an input message into different output messages" on page 1508.

Handling nulls in message maps
A message might contain fields that can carry a specific out-of-range value. This is distinct from the field being empty. Such values are termed *nil* or *null*, and the field is said to be *nillable* or *nullable*.

About this task

The logical message tree supports the concept of out-of-range values by using one of two techniques, depending on the data format:

1. For XML, the schema model allows for elements to be defined as nillable to indicate that they support an out-of-range `null` value. An XML element in a document is identified as being nilled, representing the null value, by having a **xsi:nil** attribute with the value **true**. The XMLNSC parser logical tree for a nilled element has an empty value and a child attribute set to **xsi:nil**, with the value **true**. For elements that are not defined as nillable, it is not possible to distinguish between having an empty value or the null value.
2. For other text and binary messages that are modeled with a DFDL schema or MRM message set, elements can also be defined as nillable to indicate that they support an out-of-range value. In the message bit stream, a reserved value is identified to indicate the nullable state. The DFDL and MRM parsers logical tree for a nilled element have the value set to the special value NULL.
3. For JSON messages, JSON objects and JSON arrays implicitly support the JSON null value to indicate that they support an out-of-range value. In JSON, data has a special value that is called `null` which can be set on any type of data including arrays, objects, number, and boolean types. In the Graphical Data Mapping editor, the JSON format supports null as a JSON value type for an object or an array. When a JSON message includes an object with a null value, the value in the message tree is set to NULL.

When you map nilled values, consider the behavior when you use the **Move** transform, **Custom Java**, or **Custom ESQL** to set a target.

The following table details the result when you map nilled values to different target types:

Source	Target (XML nillable)	Target (XML not nillable)	Target (Non-XML nillable)	Target (JSON object or JSON array)
Logical Message Tree XML nillable element	Target created as nilled, has xsi:nil attribute 'true', if source has xsi:nil attribute 'true'.	Target created with empty value.	Target created with NULL value, if source has xsi:nil attribute 'true'.	Target created with NULL value, if source has xsi:nil attribute 'true'.
Logical Message Tree XML element set to NULL	Target created with empty value. A Source XML having NULL value is not considered a nilled element.	Target created with empty value.	Target created with empty value. A Source XML having NULL value is not considered a nilled element.	Target created with NULL value.
Logical Message Tree non-XML	Target created as nilled. Target has xsi:nil attribute 'true', if source is NULL.	Target created with empty value.	Target created with NULL value, if source is NULL.	Target created with NULL value, if source is NULL.
Database nullable column	Target created as nilled. Target has xsi:nil attribute 'true', if source is SQL NULL.	Target created with empty value.	Target created with NULL value, if source is SQL NULL.	Target created with NULL value, if source is SQL NULL.
Custom ESQL	Target created as nilled. Target has xsi:nil attribute 'true', if return is ESQL NULL.	Target created with empty value.	Target created with NULL value, if return is ESQL NULL.	
Custom Java	Target created as nilled. Target has xsi:nil attribute 'true', if return is an MbElement with type set to "TYPE_UNKNOWN" and a value of "null" and a child element xsi:nil 'true'.	Target created with empty value.	Target created with value NULL, if return is an MbElement with type set to "TYPE_UNKNOWN" and a value of "null".	
Custom XPath expression: `iib:setNull()`	Target created as nilled. Target has xsi:nil attribute 'true', if source has xsi:nil attribute 'true'.	Target created with empty value.	Target created as nilled, has xsi:nil attribute 'true', if source has xsi:nil attribute 'true'.	Target created with JSON null value.

Accessing a user-defined policy from a graphical data map
Use the Graphical Data Mapping editor to access the properties of a user-defined policy in a static manner.

Before you begin

Complete the following tasks:

- Creating a message flow
- User Defined policy (UserDefined)

About this task

If you created a user-defined policy, and created properties for that policy, you can query those properties by using the MbPolicy.getPolicy call from a Custom Java transform in a message map.

Procedure

Complete the following steps to use Java in a Custom Java transform:

1. Create a Java class (for example, `MyPolicyAccessClass`) with the following static method:

   ```
   Static String getPolicyProperty(String policyProjectName, String propertyName)
   ```

2. Implement `getPolicyProperty` to look up the passed `policyProjectName` by using code such as in the following example.

   ```
   MbPolicy myPol = MbPolicy.getPolicy("UserDefined", policyProjectName);
   ```

3. Ensure that your method returns the value of the passed `propertyName` by using code such as in the following example.

   ```
   resultPropertyValue = myPol.getPropertyValueAsString(propertyName);
   ```

4. Create a message map.

   See “Creating message maps” on page 1378.

5. Add a Custom Java transform to the map.

   See “Mapping an element by using a Custom Java transform” on page 1496.

6. Set the following properties for the Custom Java transform.

   - Call the Java method `MyPolicyAccessClass.getPolicyProperty()`.
   - Set suitable values for the `policyProjectName` and `propertyName` parameters.

   You can map the returned `resultPropertyValue` from the output of the Custom Java transform.

Using Java in a message map
In the Graphical Data Mapping editor, you can use methods in a Java class to define a transformation, to define a conditional expression, or to change the value of an input parameter in another function.

About this task

The following guidelines apply when you use a Java class in a map:

- You can use static methods that return the appropriate type for the value of the output elements that they set.
- The values of the map input elements that are used as parameters of a static method must have the appropriate type.
- Do not define Java methods with data type overloading and then use the methods in a map. The map fails at run time with the following error: `IXJXE1039E: [ERR 0786] Multiple methods with the correct arity were found in the class '<class>' when attempting to`

```
evaluate a call to the Java extension function '{http://<class>;}<method>'
with arity of <N>.
```

- In a **Custom Java** transform, you can call Java methods that include **MbElement** data type arguments.
- When you define conditional expressions, you cannot use Java methods that include **MbElement** data type arguments.
- When you define conditional expressions, you can use Java methods that use the **DOM API**. For more information on the supported types, see "Custom Java" on page 1312.

Procedure

When you add a Java class to a map, you can use Java methods in any of the following situations:

- To use custom Java code in a Custom Java transform. For more information, see "Mapping an element by using a Custom Java transform" on page 1496 and "Processing complex or repeating elements in a Custom Java transform" on page 1498.
- To alter the value of an input argument to an XPath function.

  You alter the value in the **Properties** page of the XPath function. You can use content assist in the **Value** column that is available in the **General** tab to help you assign the elements of the source schema. For more information, see "Built-in XPath transforms" on page 1342
- To define a conditional expression. For more information, see "Defining a Java conditional expression for a transform" on page 1439.

  Note: To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a complex expression that includes XPath, Java, and any extension functions such as **iib:getUserDefinedProperty("propertyname")**.

What to do next

Deploy and test the message map. For more information, see "Troubleshooting a message map" on page 1587.

Using nested maps
You can edit some types of transforms, involving complex inputs and outputs, by entering a nested view in the Graphical Data Mapping editor.

About this task

Structural transforms control how nested elements are displayed in the Graphical Data Mapping editor. These transforms control the display of nested elements, but they do not affect the data. You can use In and Out arrow buttons and breadcrumb navigation for the nested transforms.

The following transforms can contain nested graphical data maps:

- Local map
- Join
- Append
- ForEach
- If, Else if, and Else
- Database Routine

The elements in a nested map must be mapped in order for the transform to run.

A **Local map** is a navigation aid that you use to view the map elements in a hierarchical way. A local map can have one primary input and multiple supplementary inputs, which can be either a simple type or a complex type. The output can be either a single element or an array element, but it must be a complex type. The local map does not transform data; you must specify transforms for the input and output elements in the nested map.

You can use the **Join** transform to join elements from two or more inputs. The inputs can be arrays or single elements, which can be merged using nested transforms to create a single output. The target element can be an array or single element but must be a complex type.

The **Append** transform iterates over multiple inputs in the specified order to append data. This transform takes multiple inputs of either simple or complex types. The output must be an array of either a simple type or a complex type.

The **For each** transform contains a nested map, and it iterates over one input array element (either a simple type or a complex type). The elements in the nested map must be mapped, otherwise the transform has no effect.

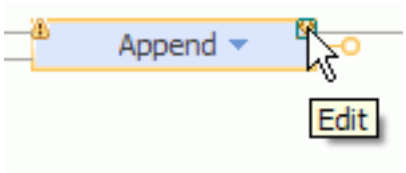
The **If**, **Else if**, and **Else** transforms enable you to control the flow of a mapping by setting conditions. If, Else if and Else operate as a group of conditional transforms, and the condition is applied to the input element of the conditional transform. If the condition is satisfied, the transform that is nested within the conditional transform is run.

The **Database Routine** transform contains a nested map to call a stored procedure or user-defined function from a database schema as input. Output from a **Database Routine** is optional, using the Return transform.

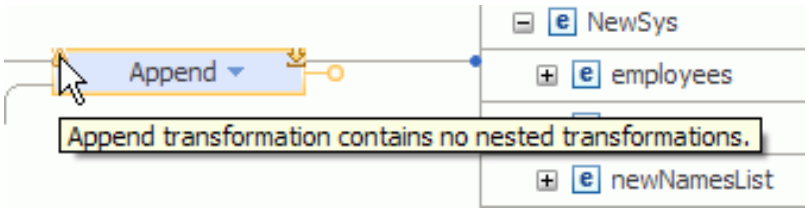
The following steps provide an example of how you can edit a nested map:

Procedure

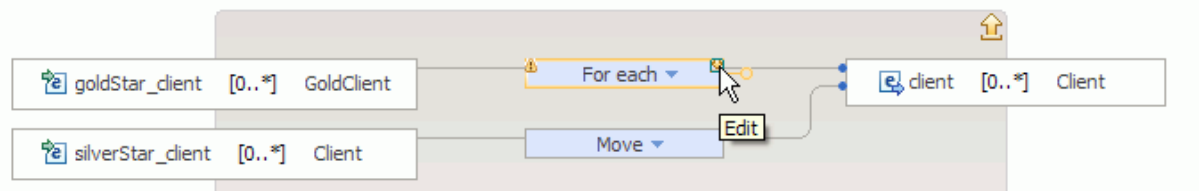
1. Some transforms, such as **Local map** and **Append**, contain nested maps. If a nested map exists, an **Edit** icon is shown on the transform. Click the **Edit** icon to edit the nested map.



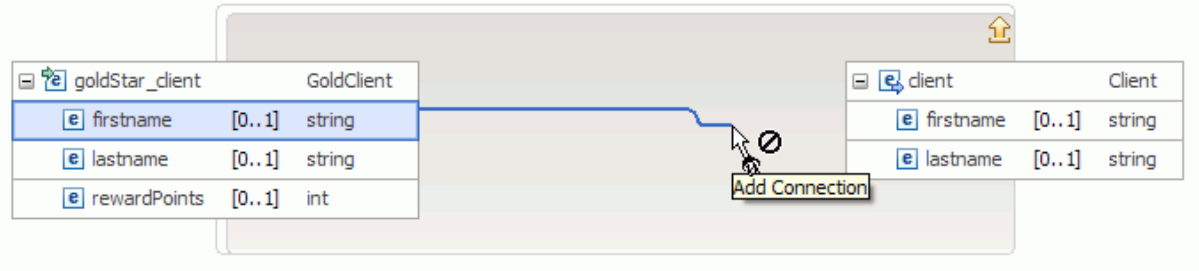
Nested maps must contain transforms, or else when the map is run, nothing happens. If a warning is displayed on the main map, you must edit the nested map.



2. When you open a nested map for some transforms such as **Append**, the nested map might also contain a nested map such as a **For each** transform:



In this case, you cannot create a mapping in the first nested map. Click **Edit** to go down another level and you can then create your mapping, as shown:



What to do next

For more information about mapping transforms, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Configuring your workspace mapping preferences

You can configure the Graphical Data Mapping editor preferences for mapping objects. You can also configure the keyboard commands that you can use when you develop message maps.

Procedure

Complete the following steps to configure your workspace:

- Optional: Configure the Graphical Data Mapping editor preferences for mapping objects. For more information, see "Setting mapping preferences for your workspace" on page 1373.

  You can set the options in the **Preferences** panel to specify the behavior of the Graphical Data Mapping editor and the mapping transforms in your workspace.
- Optional: Configure the keyboard mapping preferences in your workspace. For more information, see "Setting mapping keyboard preferences for your workspace" on page 1375.

  You can set your keyboard preferences in the Graphical Data Mapping editor by specifying your command choices in the workspace **Preferences** panel.

What to do next

Create a message map. For more information, see "Creating message maps" on page 1378.

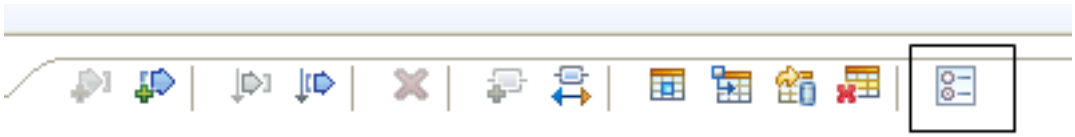
Setting mapping preferences for your workspace

You can set the options in the **Preferences** panel to specify the behavior of the Graphical Data Mapping editor and the mapping transforms in your workspace.

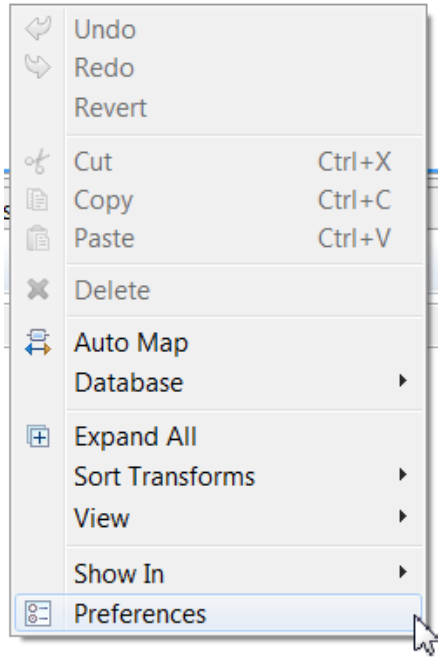
Procedure

Start the **Preferences** wizard by choosing one of the following options:

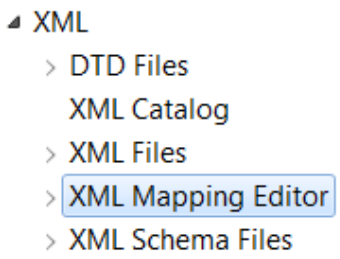
- Select the **Preferences** icon in the toolbar of the Graphical Data Mapping editor to set your preferences for the mapping.



- Right-click in the message map canvas and then select **Preferences** from the menu.

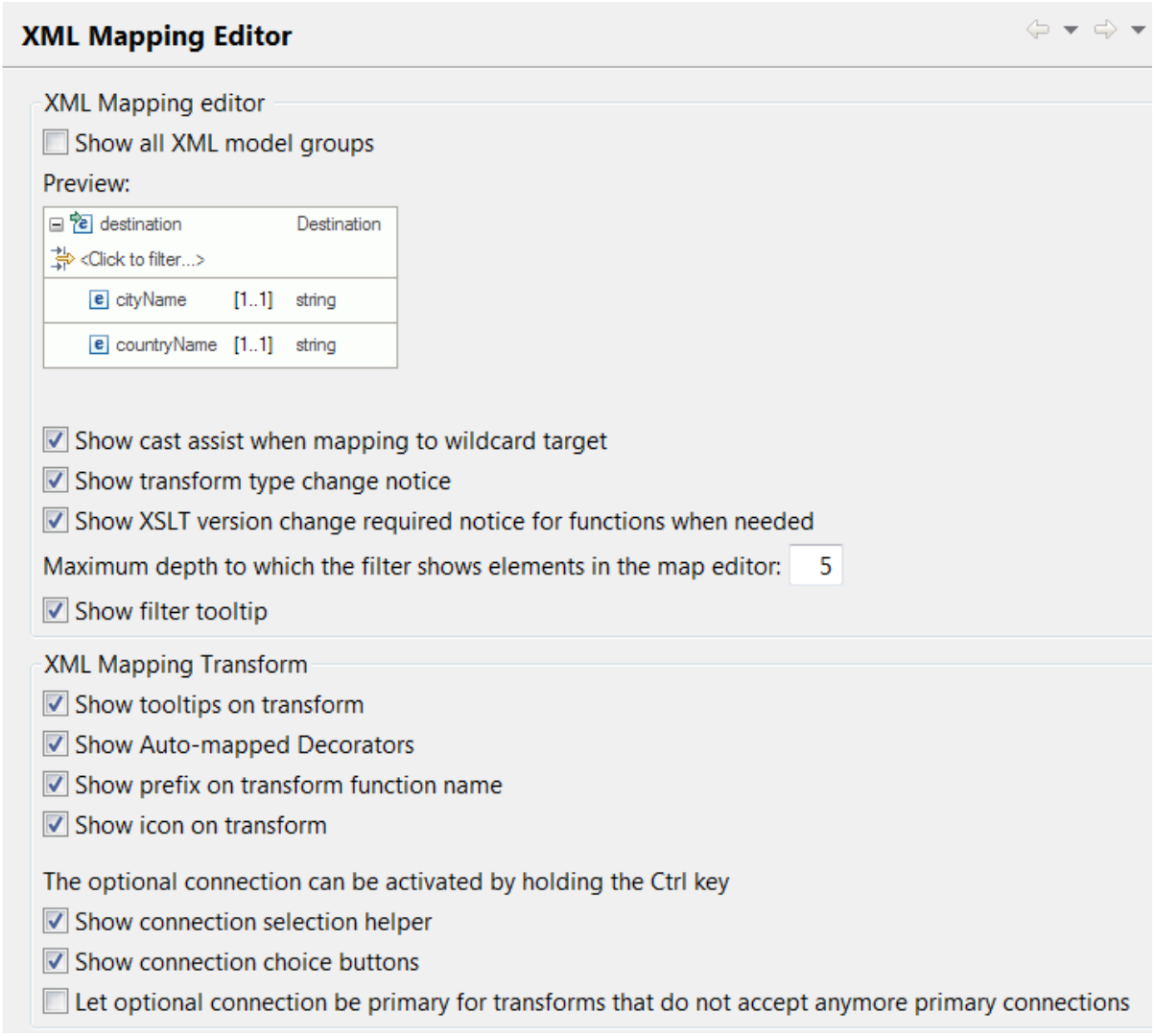


- Select **Window** > **Preferences** > **XML** > **XML Mapping Editor** in the main IBM App Connect Enterprise Toolkit menu.



Results

The **Preferences** wizard opens, and the **XML Mapping Editor** properties are displayed.



What to do next

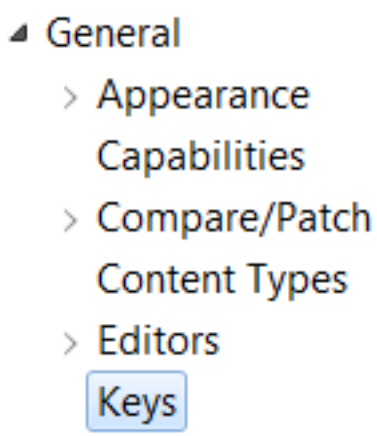
Create a message map. For more information, see “Creating message maps” on page 1378.

Setting mapping keyboard preferences for your workspace
You can set your keyboard preferences in the Graphical Data Mapping editor by specifying your choices in the workspace **Preferences** panel.

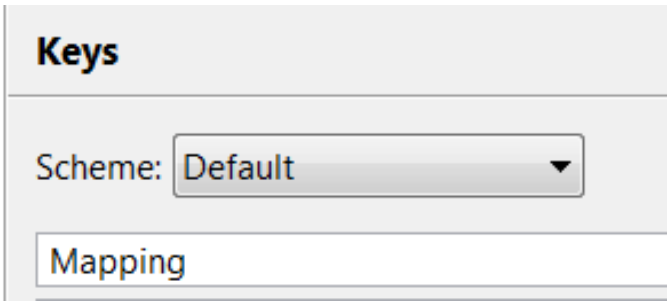
Procedure

Complete the following steps to change the default mapping keyboard commands, or to add new key combinations based on your usage specification:

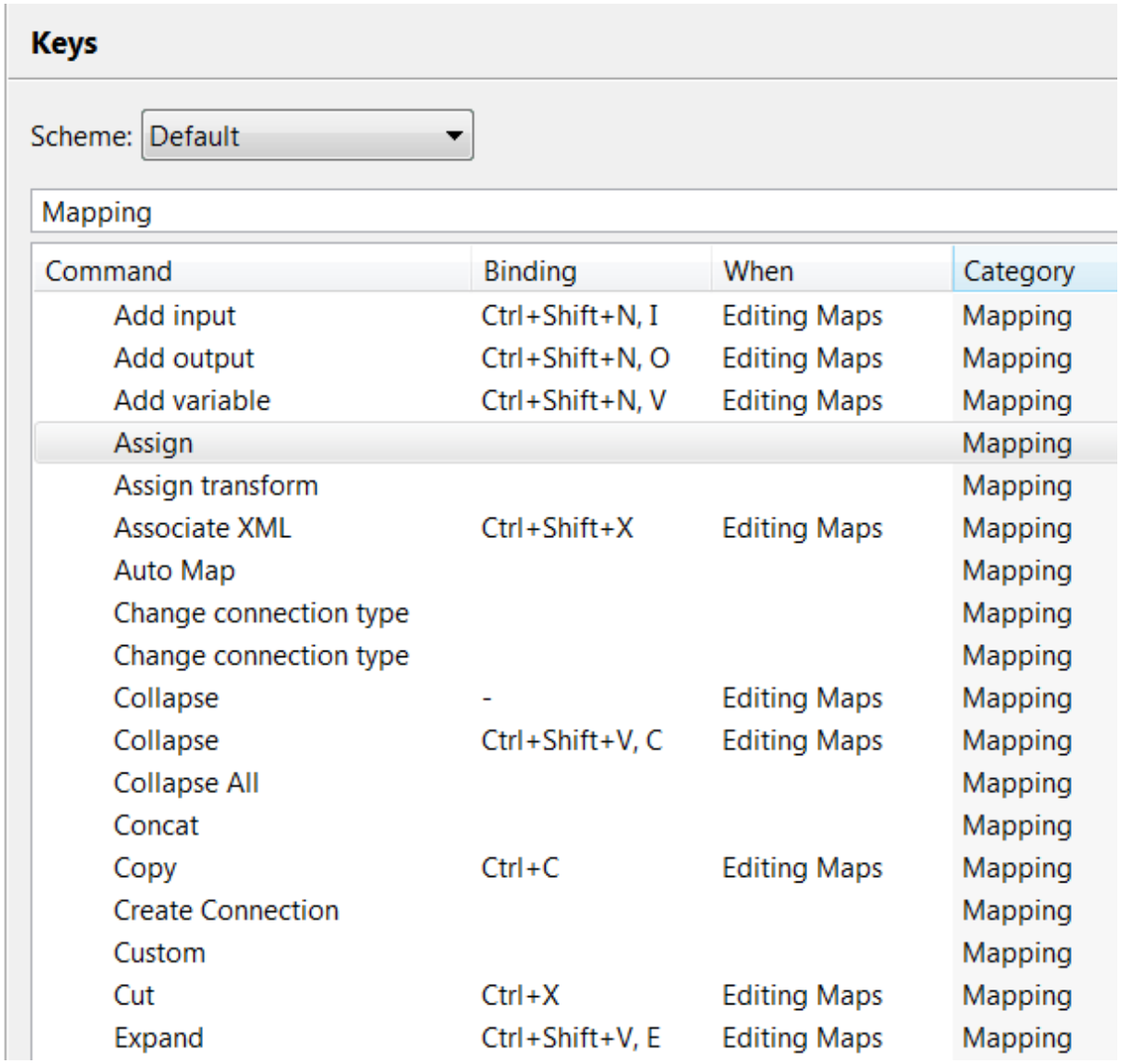
1. Select **Window** > **Preferences** > **General** > **Keys** in the main IBM App Connect Enterprise Toolkit menu.



2. Identify the entries that refer to mapping. Enter `Mapping` in the **type filter text** box.

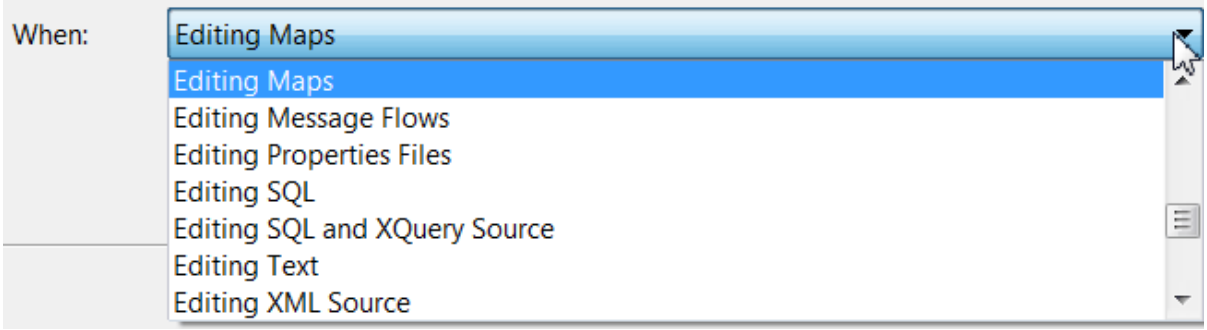


You can see the list of keys related to mapping.



3. Optional: Select a command, for example, **Assign**.

4. Optional: For each command, complete the following steps to define the key:
   a) Enter the key combination in **Binding**.
   b) Select `Editing Maps` in the field **When**.



Results

You have created a new key command for the **Assign** transform that you can use when you edit a message map in your workspace.

What to do next

Change or add new commands to specify key actions when using the Graphical Data Mapping editor.

Create a message map. For more information, see "Creating message maps" on page 1378.

Creating message maps

You can create a graphical data map, a message map, a submap, or a legacy message map to transform a message without the need to write code, providing a visual image of the transformation, and simplifying its implementation and ongoing maintenance.

About this task

A message map is the IBM App Connect Enterprise implementation of a graphical data map. It is based on XML schema and XPath 2.0 standards, with additional support for JSON schema draft 4 and Swagger 2.0. You can use a message map to graphically transform, route, and enrich a message, or implement a REST API. For more information, see "Implementing a REST API operation by using a message map" on page 1580. You can use a message map to modify data in a database system. You can use drag actions to make connections, select transforms, and build logic to transform your message data without programming.

A *submap* is a reusable form of graphical data map. Submaps enable you to use a set of mapping functions in multiple graphical data maps to transform a common set of elements in the input object to the output object. You can use a submap to reuse common data transformations. You can reuse submaps in other products that support graphical data maps.

Note: If you plan to reuse data transformations across different products, read "Guidelines for developing reusable graphical data mapping assets" on page 1273.

A *local map* is a subset of data transformations between input elements and output elements that are part of a graphical data map. You define a local map by creating a **Local map** transform in a message map. A local map is not an independent resource. There is no physical file associated with a local map. The scope of a local map is the message map. A local map is processed with the graphical data map. Local maps provide a way of breaking up a large graphical data map into nested groups of mapping elements. You can use local maps to simplify the overall graphical data map presentation. You can structure complex data transformations into nested groups that are easier to manage and implement.

A legacy message map is a message map that was created as a `.msgmap` file in earlier versions of WebSphere Message Broker (for example, in WebSphere Message Broker Version 7). In IBM App Connect Enterprise, you must convert your legacy message maps to `.map` files before you can deploy or modify

them. For more information, see “Using or converting legacy resources into message maps” on page 1588.

Procedure

Choose any of the following options to create a graphical data map:

- Create a message map to transform your messages graphically. For more information, see “Creating a message map” on page 1382.

  Use this type of map to create a graphical data map in IBM App Connect Enterprise. You can transform message assembly components, body data, and access a database to retrieve data or modify data, if needed.
- Create a graphical data map to share a map across different products that support the Graphical Data Mapping editor. For more information, see “Creating a graphical data map in the Eclipse editor” on page 1393.

  Use this type of map to create a graphical data map that you can use as a submap in IBM App Connect Enterprise, or that you can use as a main map in other products that support the Graphical Data Mapping editor. You can only transform the body of a message. You cannot access a database when you use this type of map to enrich a message or to modify database content.
- Create a submap to reuse common transformation logic. For more information, see “Creating a submap” on page 1386.

  Use this type of map to create a graphical data map that you can use as a submap in IBM App Connect Enterprise. You can only transform the body of a message. You can access a database to enrich a message or to modify database content.
- Create a local map to reduce complexity reading and managing graphical data maps. For more information, see “Creating a local map by using the Local map transform” on page 1391.

  Use this type of map to reduce the complexity of your maps and submaps. This type of map is local, and you cannot reuse it, unless you convert it to a submap.

Results

You can use the following table to identify the type of map that you must create when transforming data graphically in the Graphical Data Mapping editor:

Table 56. Types of map based on design requirements

	Recommended use	Type of resource	Supported in IBM App Connect Enterprise
Message map	Transform messages graphically	`.map` file	Yes
Submap	Reuse of common data transformations	`.map` file	Yes
Local map	Reduce complexity reading and managing a Message map	No file. It is embedded within a Message map	Yes
Graphical data map	Share graphical data maps across different software products	`.map` file	Yes

Table 56. Types of map based on design requirements (continued)			
	Recommended use	**Type of resource**	**Supported in IBM App Connect Enterprise**
Legacy message map	Solutions migrated from earlier versions of WebSphere Message Broker	`.msgmap` file	These files must be converted into `.map` files before they can be used in IBM App Connect Enterprise. (See note below.)

Note: Legacy message maps must be converted into `.map` message map format before they can be deployed or modified by IBM App Connect Enterprise.

You can see the message map in the **Application Development** view displayed under a **Maps** category, and organized by namespace.

What to do next

After you create a graphical data map, edit the map, and define transformations between the input message and the output message. For more information, see "Editing message maps" on page 1394.

Choosing the project structure when you create a message map
In IBM App Connect Enterprise, you can create a message map in an application project, a static library project, a shared library project, or an integration project.

About this task

You can use any of the following types of resources in a message map:

- XML schema models
- DFDL data models
- User-defined models
- ESQL routines
- Java code
- Submaps
- JSON schema files
- REST API swagger document

Many of these resources can be defined in the same project as the map or in a different one; however, for JSON schema and REST API swagger documents, the map must be in the same project container. When a resource is defined in a different project to the map, the project where the map is defined must reference it.

XML schema models and DFDL data models

When you define your message model by using a global XML schema or a DFDL model, consider the following rules:

- You can have the map and any schema definitions that are used in the map in the same project. The project can be a static library project, a shared library project, or an application project.
- You can have any schema definition that is used in the map in a static library project and the map in an application project.
- You can have any schema definition that is used in the map in a shared library project and the map in a different shared library project.

  Note: The XML schema files that are defined in the application project cannot access XML schema files that are defined in a shared library.

- You can have any schema definition that is used in the map in a shared library and the map in an application project.
- You can have any schema definition that is used in the map in a static library project and the map in an integration project.

User-defined models

When you use user-defined elements in your message model and you define these elements by using a global XML schema, consider the following rules:

- You can have the map and any schema definitions that are used in the map in the same project. The project can be a static library project, a shared library project, or an application project.
- You can have any schema definition that is used in the map in a static library project and the map in an application project.
- You can have any schema definition that is used in the map in a shared library project and the map in a different shared library project.
- You cannot have the schema definitions that are used in the map in a shared library and the map in an application project. The Graphical Data Mapping editor reports error **CWMSL209E** if you break this rule.
- You can have any schema definition that is used in the map in a static library project and the map in an integration project.

Submaps

When you use **submaps** in a message map, consider the following rules:

- You can have the map and submap in the same project. The project can be a static library project, a shared library project, or an application project.
- You can have a submap that is used in the map in a static library project and the map in an application project.
- You cannot have a submap that is used in the map in a shared library and the map in an application project. The Graphical Data Mapping editor reports error **CWMSL209E** if you break this rule.
- You can have a submap that is used in the map in a static library project and the map in an integration project.

Java code

When you define transformations in your map that use Java code, you can define a map in a static library project, a shared library project, an application project, or an integration project. You can define the Java code in one or more Java projects. Your map project must reference these Java projects.

Note: When the map uses Java code available in a shared library, you must create the map in the shared library project where the Java code is available.

ESQL routines

When you define transformations in your map that use ESQL code, consider the following rules:

- You can have the map and the ESQL code that is used in the map in the same project. The project can be a static library project, a shared library project, an application project, or an integration project.
- You can have the ESQL code in a static library project and the map in an application project, that references this static library.
- You can have the ESQL code in a shared library project and the map in a different shared library project that has a reference to it.

  Note:

  – When you store ESQL files in a shared library, you must place the ESQL files inside a schema that is not the default empty schema.

  – Each shared library can have one or more ESQL files in one or more ESQL schemas.

  – The schema in a shared library must be unique:

  - An application or a shared library cannot reference two shared libraries that have ESQL in the same schema.
  - An application with ESQL in schema A cannot reference a shared library with any ESQL in schema A.
- You can have the ESQL code in a static library project and the map in an integration project.

JSON schema and REST API Swagger models

When you define your message model by using JSON schema or REST API Swagger models, ensure that you are aware of the requirements specified in "JSON schema requirements for message maps" on page 1549.

Procedure

Choose one of the following project structures based on the resources that are used in the map if you want to update a map without redeploying your application:

- If the map includes XML schemas, DFDL models, or both, create the map and associated resources in one or more shared libraries. Reference the shared library project that contains the map from the application project.
- If the map includes XML schemas, DFDL models, and user-defined elements, create the map and the user-defined elements in the same shared library project. You can use the same or different shared library projects for the other resources. Reference the shared library project that contains the map from the application project.
- If the map includes XML schemas, DFDL models, and transformations that use ESQL code, create the map and the ESQL files in shared library projects. You can use the same or different shared library projects for the other resources. You must define the ESQL files within a schema. The schema names that are used by all the projects that contain ESQL files must be unique. Reference the shared library project that contains the map from the application project.
- If the map includes XML schemas, DFDL models, and transformations that use Java code, create the map in the shared library project where the Java code is available. Reference the Java project from the shared library project.
- If you want to map between different versions of an XML schema, where both versions might declare the same elements and types and cannot coexist in one application, you can build one shared library that contains version 1 of an XML schema file, and another shared library that contains version 2. You can then build a map that references the XML schema files in the shared libraries and maps between the two versions. At deployment time, separate models are built for each of the shared libraries. Therefore, no clash results from the duplicated elements and types.
- If the map includes JSON schema, create the map and JSON schema in a single shared library. Reference the shared library project that contains the map from the application project that contains the mapping node.
- If the map includes REST API, and optionally JSON schema (or both), create the map and JSON schema in the REST API project.

Creating a message map

You can create message maps either in the **Application Development** view of the IBM App Connect Enterprise Toolkit, or from a Mapping node in a message flow.

Before you begin

Before you create a message map, you must complete the following tasks:

1. Create an application, a library, an integration service, or an Integration project, as described in the following topics:
   - Creating an application
   - Creating a library
   - Creating an integration project
   - Creating a REST API. For information about the steps involved in completing this task, see “Implementing a REST API operation by using a message map” on page 1580 .
2. Create a message flow. For more information, see Creating a message flow.
3. Define the message flow content that includes a Mapping node. For more information, see Defining message flow content.

About this task

A message map is a graphical data map.

The physical representation of a message map is a `.map` file.

Procedure

You can use any of the following methods to start the New Message Map wizard and then create a message map:

- Create a message map in the **Application Development** view. For more information, see “Creating a message map in the Application Development view” on page 1383.
- Create a message map from a Mapping node in a message flow. For more information, see “Creating a message map from a Mapping node” on page 1385.

Results

A message map is created as a `.map` file.

In the **Application Development** view, the message map is displayed under a **Maps** category. Maps are organized by namespace.

What to do next

Edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see “Mapping input to output elements manually” on page 1418.

*Creating a message map in the **Application Development** view*
You can create a message map for use in your message flows in the **Application Development** view with messages as input and output objects. Data from database tables can also be used as input to the message map.

Before you begin

Before you create a message map, you must complete the following tasks:

1. Create an application, a library, an integration service, or an Integration project, as described in the following topics:
   - Creating an application
   - Creating a library
   - Creating an integration project
   - Creating a REST API. For information about the steps involved in completing this task, see “Implementing a REST API operation by using a message map” on page 1580 .

2. Create a message flow. For more information, see "Creating a message flow" on page 574.
3. Define the message flow content that includes a Mapping node. For more information, see "Defining message flow content" on page 614.

Procedure

To create a message map in the **Application Development** view, complete the following steps:

1. Choose one of the following options to start the New Message Map wizard:
   - In the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Message Map**.
   - In the **Application Development** view, right-click the application, library, integration service, or Integration project where you want to create the message map. Then, click **New** > **Message Map**.

   The New Message Map wizard opens.
2. On the **Specify a new message map file** pane, select **Message map called by a message flow**.

   A message map is created that can be accessed from a Mapping node. A message map can contain components of a message body such as global elements and global types. A message map contains Properties, message headers, or the local environment.
3. Specify the **Container**, **Map name**, and broker schema for the message map. Click **Next**.

   If your message map is likely to be used by multiple solutions, store it in a shared library.
4. On the **Select map inputs and outputs** pane, select your input type:
   - If you want to use an XML, DFDL, or JSON schema-defined element, expand the list of available input objects, and select the input objects that you want to use as inputs to the message map.

     If necessary, use the **Filter map input names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
   - If you want to use SOAP, JSON, or BLOB messages as input, expand **IBM supplied message models**, and select the message model type.
5. Select your mapping output type:
   - If you want to use an XML, DFDL, or JSON schema-defined element, expand the list of available output objects, and select the output objects that you want to use as outputs for the message map.

     If necessary, use the **Filter map output names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
   - If you want to use SOAP, JSON, or BLOB messages as input, expand **IBM supplied message models**, and select the message model type.
6. Optional: If you are creating a **Message map called by a message flow node**, click **Next** to specify the **Output domain** for the message map.
7. Click **Finish** to create the message map.

Results

The new message map is created, and the Graphical Data Mapping editor opens with the selected sources and targets.

The message map is created as a `.map` file.

In the **Application Development** view, the message map is displayed under a **Maps** category. Maps are organized by namespace.

What to do next

Edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see "Editing message maps" on page 1394.

Creating a message map from a Mapping node
You can use a Mapping node to create a message map with messages as input and output objects. Data from database tables can also be used as input to the message map.

Before you begin

Before you create a message map, you must complete the following tasks:

1. Create an application, a library, an integration service, or an Integration project, as described in the following topics:
   - Creating an application
   - Creating a library
   - Creating an integration project
   - Creating a REST API. For information about the steps involved in completing this task, see "Implementing a REST API operation by using a message map" on page 1580 .
2. Create a message flow. For more information, see Creating a message flow.
3. Define the message flow content that includes a Mapping node. For more information, see Defining message flow content.

Procedure

To create a graphical data mapping (`.map`) file from a Mapping node:

1. In the Integration Development perspective, open your message flow.
2. Double-click a Mapping node that does not have a message map associated with it, or right-click the Mapping node and click **Open Map**.

   The New Message Map wizard opens.
3. On the **Specify a new message map file** pane, the type of message map that you want to create is selected as **Message map called by a message flow node**. This is a message map that can be accessed from a node.
4. Specify the **Container**, **Map name**, and **broker schema** for the message map, or use the values that have been entered for you by the wizard. Click **Next**.

   If your message map is likely to be used by multiple solutions, store it in a shared library. If you are using a JSON schema model, the container must be the shared library in which your JSON schema files are stored.
5. On the **Select map inputs and outputs** pane, select your input type:
   - If you want to use a DFDL, XML, or JSON schema-defined element, expand the list of available input objects, and select the input objects that you want to use as inputs to the message map.

     If necessary, use the **Filter map input names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
   - If you want to use SOAP, BLOB, or JSON messages as input (and you have no JSON schema), expand **IBM supplied message models**, and select the message model type.
6. Select your mapping output type:
   - If you want to use a DFDL, XML, or JSON schema-defined element, expand the list of available output objects, and select the output objects that you want to use as outputs for the message map.

     If necessary, use the **Filter map output names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
   - If you want to use SOAP, BLOB, or JSON messages as input, (and you have no JSON schema), expand **IBM supplied message models**, and select the message model type.
7. Optional: On the **Select the domain to create the output** pane, specify the **Output domain** for the message map (if it is not the same as the output type selected in step "6" on page 1385).
8. Click **Finish** to create the message map.

Results

The new message map is created, and the Graphical Data Mapping editor opens with the selected inputs and outputs.

A message map is created as a `.map` file.

In the **Application Development** view, the message map is displayed under a **Maps** category. Maps are organized by namespace.

What to do next

Edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see "Editing message maps" on page 1394.

Creating a submap
You can create a submap from scratch or after you define a **Submap** transform between an input object and an output object in a message map. You can also convert a local map into a submap.

About this task

You can use a submap to reuse common data transformations between input objects and output objects.

Procedure

Choose one of the following methods to create a submap:

- Create a submap from scratch.

  Use this method to create a common transformation between an input object and an output object.

  For more information, see "Creating a submap in the Application Development view" on page 1386.
- Create a submap by using the Graphical Data Mapping editor. For more information, see "Creating a submap by using the Submap transform" on page 1387.

  Use this method when you have a message map where you have identified a common transformation between an input object and an output object.
- Create a submap by converting a local map to a submap. For more information, see "Converting a local map into a submap" on page 1388.

  Use this method when you have a local map defined in your message map that you have identified as a common transformation between an input object and an output object.

What to do next

Edit the submap and define the transformation logic between the input and output elements in the submap. For more information, see "Specifying a transform (mapping operation)" on page 1430.

*Creating a submap in the **Application Development** view*
You can create a submap for use in your message flows in the **Application Development** view with messages as input and output objects. Data from database tables can also be used as input to the message map.

About this task

A submap enables you to use the same piece of mapping function in multiple message maps.

Procedure

Complete the following steps to create a submap in the **Application Development** view:

1. Choose one of the following options to start the New Message Map wizard:
   - In the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Message Map**.
   - In the **Application Development** view, right-click the application, library, integration service, or Integration project where you want to create the message map. Then, click **New** > **Message Map**.

   The New Message Map wizard opens.
2. On the **Specify a new message map file** pane, select **Submap called by another map**.

   A submap is created that can be referenced from another message map. A submap can contain components of a message body such as global elements and global types. A submap does not contain Properties, message headers, or the local environment.
3. Specify the **Container**, **Map name**, and broker schema for the submap. Click **Next**.
4. On the **Select map inputs and outputs** pane, expand the list of available input objects, and select the input objects that you want to use as input to the submap.

   If necessary, use the **Filter map input names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
5. Expand the list of available output objects, and select the output object that you want to use as output for the submap.

   If necessary, use the **Filter map output names** field to filter what is shown in the list of available objects. Each object in the list is displayed in the form: `objectname {namespace}`.
6. Click **Finish** to create the submap.

Results

The submap is created, and the Graphical Data Mapping editor opens with the selected sources and targets.

The submap is created as a `.map` file.

You can see the submap in the **Application Development** view displayed under a **Maps** category, and organized by namespace.

What to do next

Edit the submap, and define transformations between the input message assembly and the output message assembly. For more information, see “Editing message maps” on page 1394.

*Creating a submap by using the **Submap** transform*
Create a submap by using the Graphical Data Mapping editor. Create a submap by defining a **Submap** transform between an input object and an output object in a message map.

About this task

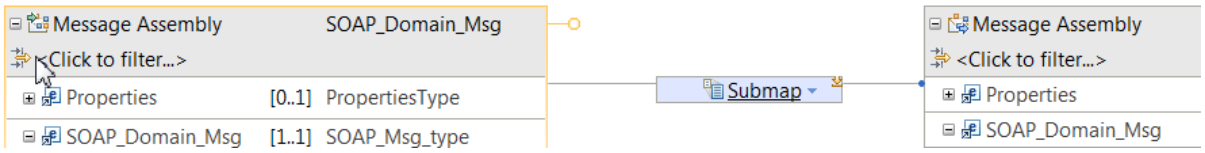
A submap enables you to use the same piece of mapping function in multiple message maps.

Procedure

Complete the following steps to create a submap:

1. Create a connection between global input and output elements in a message map, and then select the **Submap** transform on the connection:

   For example:



2. In the Properties view of the Submap transform, create a submap to use an existing submap:
   - To create a submap, click **New**. The New Message Map wizard opens.
   - Use an existing submap. Click **Browse**. A dialogue box will display the submaps that are available. Then, select a submap and click **OK**.
3. On the **Specify a new message map file** pane, the type of map that you want to create is selected as **Submap called by another map**. This is a message map that can be referenced from another message map. This is known as a submap and can contain components of a message body such as global elements and global types. A submap does not contain Properties, message headers, or the local environment tree. Click **Next**.
4. On the **Select map inputs and outputs** pane, the input and output objects of the submap have been pre-selected.
5. Click **Finish**.

   The new submap is displayed in the Graphical Data Mapping editor, and you can edit it in the same way that you would edit any graphical data map. For information about how to edit maps, see "Editing message maps" on page 1394.

Results

The new submap is displayed in the Graphical Data Mapping editor.

What to do next

Edit the submap. For more information, see "Editing message maps" on page 1394.

Converting a local map into a submap
Use the Graphical Data Mapping editor to convert a local map into a submap by using the **Refactor to submap** function.

About this task

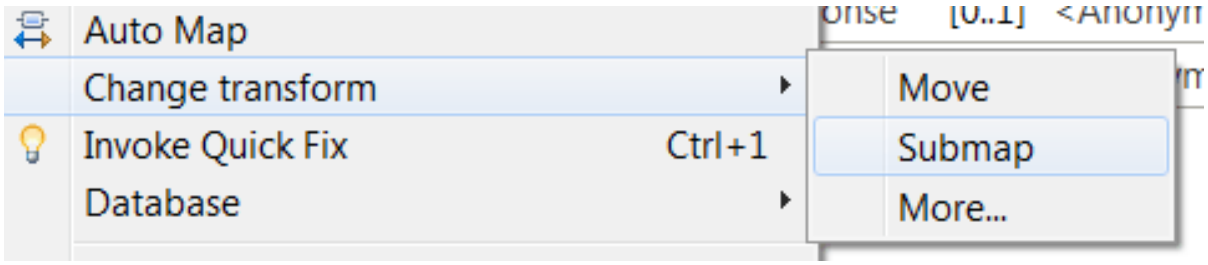
You can convert a local map into a submap so that the transformation logic can be reused by other graphical data maps.

To convert a local map into a submap, the following conditions must be true:

- The input and output elements to the local map must be global elements.
- There must be at least one transform configured in the local map.

If you want to convert a local map to an existing submap, you cannot use the **Refactor to submap** function. You must change the transform from a **Local map** transform to a **Submap** transform.

Note: If you have global elements defined as input and output objects to the local map, and you have not defined any transformations between the input and output objects, you cannot convert a local map into a submap. To create the submap, you can change the **Local map** transform for the **Submap** transform. The following figure shows the menu option that you can use to create a submap instead of a local map:

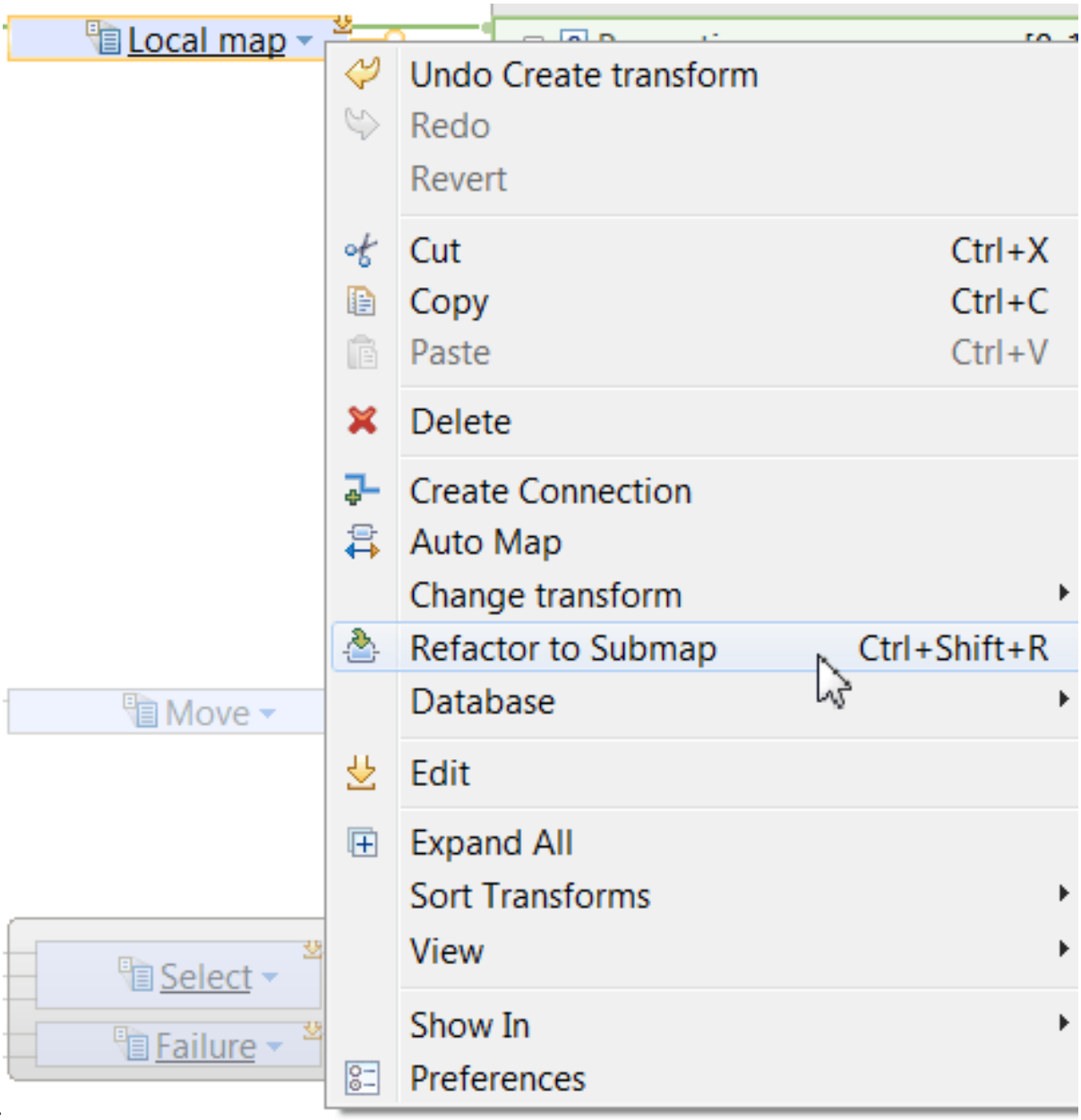


Procedure

To convert a local map into a submap, complete the following steps:

1. Right-click the **Local Map** transform in the graphical data map, and then select **Refactor to submap**.

   The following figure shows the menu that opens when you right-click the **Local map**



   transform:

   The New Message Map wizard opens.

2. On the **Specify a new message map file** pane, the type of map that you want to create is selected as **Submap called by another map**. This is a message map that can be referenced from another message map. This is known as a submap and can contain components of a message body such as global elements and global types. A submap does not contain Properties, message headers, or the LocalEnvironment. Click **Next**.
3. On the **Select map inputs and outputs** pane, the input and output objects of the submap have been pre-selected.
4. Click **Finish**.

   The new submap is displayed in the Graphical Data Mapping editor, and you can edit it in the same way that you would edit any graphical data map. For information about how to edit maps, see "Editing message maps" on page 1394.

Results

A submap is created, containing all the mappings from the local map.

What to do next

Edit the submap. For more information, see “Editing message maps” on page 1394.

Creating a message map programmatically
Use the Graphical Data Map Specification Language to create a message map programmatically.

About this task

The file `msl.xsd` provides the XML schema that describes the *Graphical Data Map Specification Language*, also known as **MSL**. This file is available in the Graphical Data Mapping component Version 1040, and later versions.

You can find the `msl.xsd` file inside the `com.ibm.msl.mapping.api_7.5.0.jar` jar file.

In IBM App Connect Enterprise, you can find the jar file in any of the following directories:

- In a Windows platform installation: `C:\Program Files\IBM\IIB\10.0.0.0\server\ct\lib\`
- In a Linux platform installation: *`install_dir`*`/iib-10.0.0.0/server/ct/lib/`

You can create a map programmatically when you have meta-data that defines the transformation logic that needs to be applied to an input message to produce the output. For example, you can write a JAXB based program that builds the map using JAXB classes generated from the provided MSL schema. The program reads XML data that defines the input elements to be mapped and the output elements.

Procedure

You must perform the following steps to programmatically generate a map file:

1. Create a template map in the Graphical Data Mapping editor.

   Deploy and test your template map in the run time to confirm that the message transformation is correct.
2. Inspect the MSL XML content in the template map.

   Use the MSL schema to identify the mapping constructs and the definition of the points of variation for the template map.
3. Develop the scripts and programs to generate the MSL XML for the new maps that you plan to generate programmatically.

   When you use a development approach based on JAXB, the bindings file `msl_jaxb_bindings.xml` available in the `com.ibm.msl.mapping.api_7.5.0.jar` file provides the minimal required bindings.
4. Validate the syntax of each generated map file (`.map`) against the provided MSL schema `msl.xsd`.
5. Import each generated map file into your development environment. Then, check that all the referenced resources, such as `xsd` files, are imported into the relevant project types. Ensure the relevant builder is invoked to semantically validate each generated map file. Also, check using the Graphical Data Mapping editor that the transforms in your generated map are correct and free of error and warnings.
6. Package and deploy into the run time your programmatically generated maps. Then, test your application to confirm that the message transformation is correct.

   Check that each generated map file is built into the relevant deployment artifact.

What to do next

Deploy and test the message map. For more information, see “Troubleshooting a message map” on page 1587.

*Creating a local map by using the **Local map** transform*
Create a local map by using the Graphical Data Mapping editor. Create a local map by defining a **Local map** transform between an input object and an output object in a message map.

About this task

You can use a local map to break up a large map into nested groups of mapping elements and process the complex elements of the whole data object.

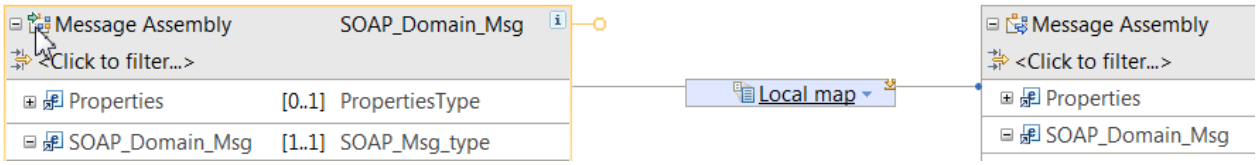
Local maps are a partial view of a larger map, rather than separate files.

A local map has only one element as input (either a simple type or a complex type), which can contain nested elements. The output can be either a single element or an array element, but it must be a complex type.

Procedure

Create a connection between one input element and one output element in a message map, and then select the **Local map** transform on the connection:

For example:



Results

The local map opens within the main map in the Graphical Data Mapping editor.

What to do next

Edit the local map. For more information, see “Editing message maps” on page 1394.

Converting a submap into a local map
You can use the Graphical Data Mapping editor to change a submap into a local map by using the **Refactor from submap** function.

About this task

You can customize the logic of a submap in a graphical data map, without altering the submap logic, by converting the submap into a local map.

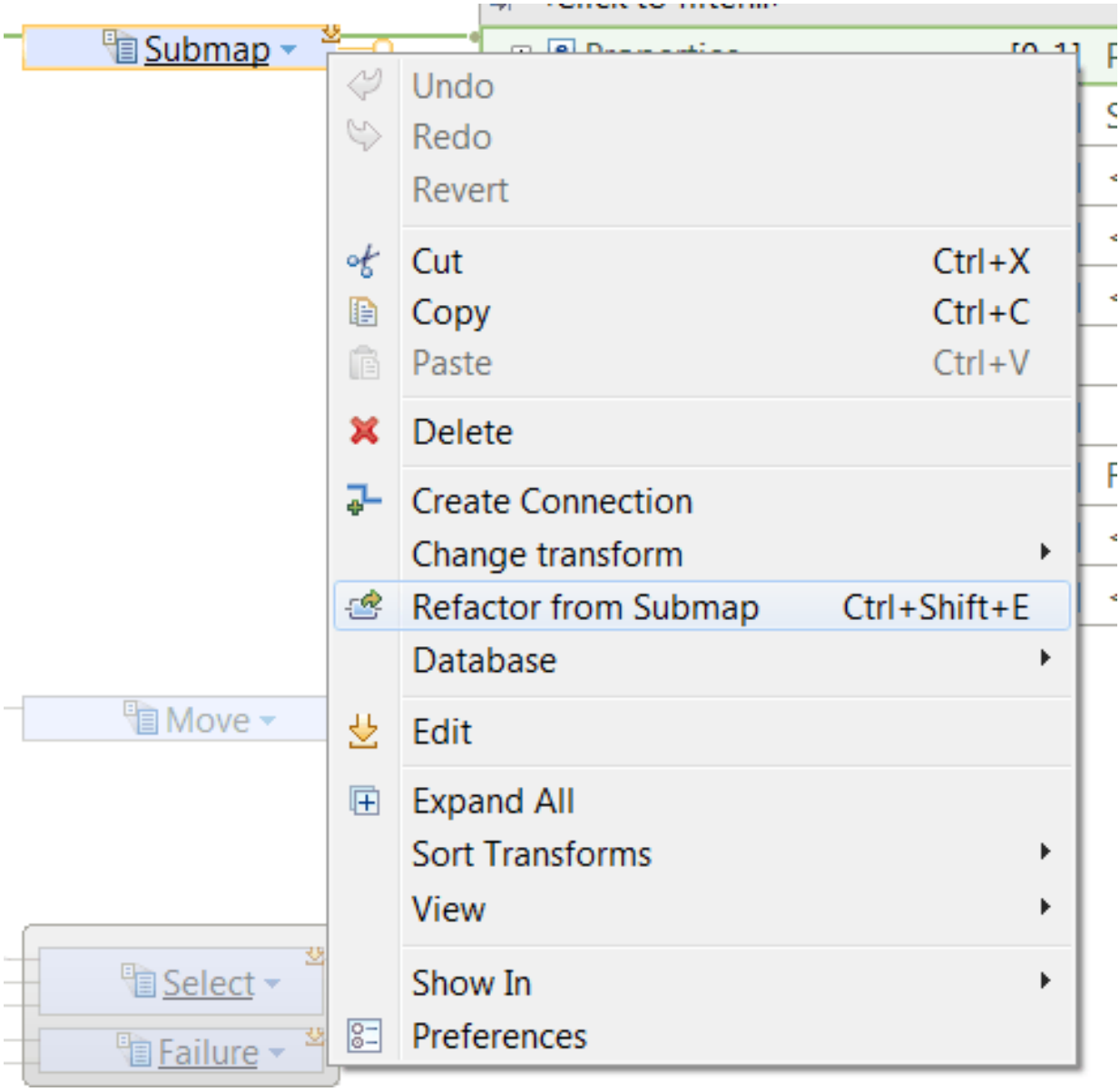
For example, you have a submap that contains common transformation logic that cannot be deleted. The submap is used by multiple graphical data maps. For one application, you need to specialize the submap, so rather than starting from scratch and creating a graphical data map, you can pull the submap logic into a main map by converting the submap to a local map. The action leaves the submap in place for ongoing use in other maps.

Procedure

To change a submap into local map, complete the following steps:

1. Right-click the Submap transform in the graphical data map, and then select **Refactor from submap**.

   The following figure shows the menu that opens when you right-click the **Submap** transform:



2. Optional: Delete the map file corresponding to the submap that you have converted to a local map if the submap is not being used anywhere else in your solutions. For more information, see "Deleting objects and transforms" on page 1444.

Results

A local map is created, containing all the mappings that were included in the original submap.

What to do next

Continue editing the graphical data map. For more information, see "Editing message maps" on page 1394.

Creating a graphical data map in the Eclipse editor
Create a graphical data map in the Eclipse editor to transform data graphically so that you can reuse it in other products that support graphical data maps.

Before you begin

Before you create a reusable message map, you must complete the following tasks:

1. Create an application, a library, an integration service, or an Integration project, as described in the following topics:
   - Creating an application
   - Creating a library
   - Creating an integration project
2. Create a message flow. For more information, see Creating a message flow.
3. Define the message flow content that includes a Mapping node. For more information, see Defining message flow content.

If you plan to reuse the graphical data map in other products, read "Guidelines for developing reusable graphical data mapping assets" on page 1273.

About this task

In IBM App Connect Enterprise, you can create a graphical data map as a reusable message map. You handle a graphical data map as a submap.

Procedure

To create a graphical data map in the Eclipse editor, complete the following steps:

1. Create a new graphical data map, by selecting **File** > **New** > **Other** > **XML** > **Data map**. Click **Next**.

   The New XML Mapping wizard opens.
2. Select the parent folder for the new graphical data map and specify a file name. By default the new file is called `NewMAP.map`. Click **Next**.

   The Input and Output Roots pane is displayed.
3. Select the input and output root files by clicking **Add** and then browsing to the required schema files.

   Note: If you choose not to select the input and output schemas at this point, you can select **Next**, and the Graphical Data Mapping editor is displayed with no input or output objects. You can then select your input and output objects in the Graphical Data Mapping editor by using the **Add an input object** and **Add an output object** icons.

   When you have specified your input and output objects, they are shown in the Graphical Data Mapping editor with the input object on the left side of the canvas, and the output object on the right side:
4. Click **Next**.
5. Click **Finish**.

What to do next

Edit the graphical data map, and define transformations between the input message and the output message. For more information, see "Editing message maps" on page 1394.

Editing message maps

You edit a message map in the Graphical Data Mapping editor.

Before you begin

Create a message map using the Graphical Data Mapping editor. For information about how to do this, see "Creating a message map" on page 1382.

About this task

In the Graphical Data Mapping editor, you create a map, you define the input and output message models to the map, and the transformations that must be applied to create the output message with values from the input elements. When your input or output message model includes an **xsd:any** element, you must qualify this extension point by using the **Cast** function or the **Add user defined** function prior to defining transforms.

You use the Graphical Data Mapping editor to map (or connect) elements of input objects to elements of output objects. Then, for each mapping, you can create a **transform**, which performs an action on the data of the input element and puts the result in the output element. The input objects are on the left side in the Graphical Data Mapping editor, and the output objects are on the right.

You can define XPath conditional expressions on a transform. This expression determines whether the transform is applied. Use content assist to set the required parameters of the expression. For more information, see "Using content assist (Mapping syntax)" on page 1443.

Procedure

You can do any of the following editing tasks in the Graphical Data Mapping editor:

1. Optional: Configure the general properties of a message map to define the Java, ESQL, and XSD resources that the map can refer to; to define your solution XML namespaces, and and to add any documentation. For more information, see "Configuring the general properties of a message map" on page 1395.
2. Redefine `xsd:any` elements in your message.
   a) Use the **Cast** function to redefine parts of the input or output model in a graphical data map by using a schema model. For more information, see "Mapping xsd:any on an input or output message" on page 1397. For more information, see "Mapping xsd:any on an input or output message" on page 1397.
   b) Use the **Add user defined** function to define dynamically the input and output message models when you do not have a schema model available for any of them. For more information, see "Defining user-defined elements" on page 1408.
3. Map the input and output elements in any of the following ways:
   a) Map elements manually. For more information, see "Mapping input to output elements manually" on page 1418.

      Use this method to select the input and output elements, create connections between them, and specify the required transforms.
   b) Map elements automatically. For more information, see "Mapping input to output elements automatically" on page 1421.

      Use the Auto map wizard (automap) to map elements by examining the names of input and output elements to create the mappings.
4. Specify a transform, also known as a mapping operation. For more information, see "Specifying a transform (mapping operation)" on page 1430.
5. Configure the properties of a transform. For more information, see "Configuring the properties of a transform" on page 1431.

6. Delete objects and transforms. For more information, see “Deleting objects and transforms” on page 1444.

What to do next

Configure the input and output message assembly in a message map by using the Graphical Data Mapping editor. For more information, see “Advanced editing in a message map” on page 1445.

Configuring the general properties of a message map
You can configure the general properties of a message map in the Graphical Data Mapping editor.

About this task

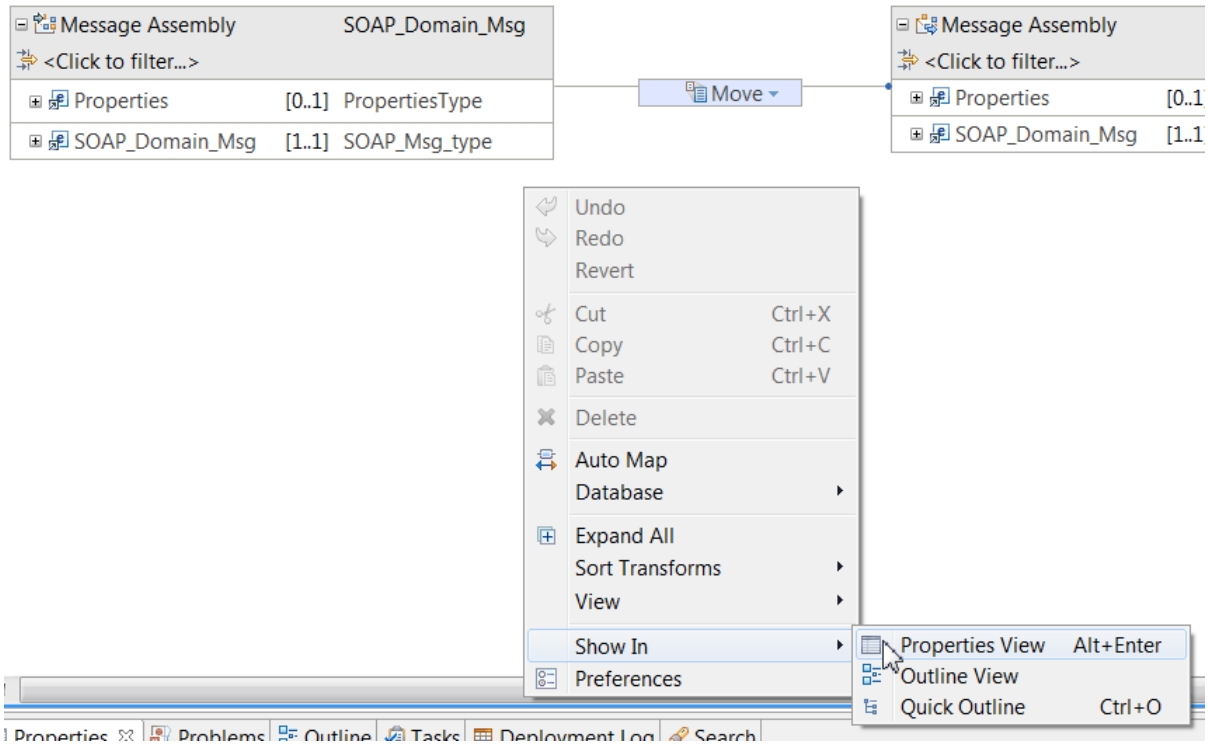
You can optionally configure the general properties of a message map:

- You can add documentation.
- You can configure prefixes for your XML namespaces.
- You can explicitly define references to Java, ESQL, and XSD resources that the map can refer. Note the Graphical Data Mapping editor will normally add these automatically while editing.

Procedure

Complete the following steps to configure the general properties of a map:

- Open the **Properties** tab of a map by using any of the following methods:
  - Select a map by focusing on the map canvas, rather than on an individual item in the map. Then, select the **Properties** tab.
  - Right-click the map canvas, and then select **Show in** > **Properties view**.



  - Use the appropriate keyboard combination, which, by default, is `Alt+Enter`.

    You can configure these shortcut keys by selecting **Window** > **Preferences** > **General** > **Keys**.

- In the **Properties** tab, add, modify, or remove resources:
  a) Select the **General** tab to view the namespace where the map is available.
  b) Select **Java Imports** to add or remove Java classes.

     If you add a Java call in the **General** tab of the **Properties** page of a **Custom Java** transform, an import is automatically added in the map to refer to the package qualified Java class, and a prefix based on the class name is added. All the **public static methods** in this Java class are then available in content-assist when building expressions in other transforms.

     If you only plan to use Java in XPath expressions on other transforms, you must first manually import the Java class into the map. You configure the **Java imports** tab in the **Properties** page of the map.

     For more information, see "Mapping an element by using a Custom Java transform" on page 1496.
  c) Optional: Select **ESQL Imports** to add or remove ESQL files that **Custom ESQL** transforms can refer to in the message map.

     When you include a **Custom ESQL** transform, an import is added to refer to the ESQL file, defining a prefix based on the file name. If you need to use custom ESQL only in condition or filter expressions, you can add ESQL imports to your ESQL file so that the applicable modules are available through content assist when you are composing an expression.
  d) Select **Scope** to add or remove XSD files that you can refer to in the message map.
  e) Select **Cast** to obtain the list of input and output wildcard elements that are cast to a specific type or global element in the message map. You can remove any entry that is not required anymore.
  f) Select **Namespaces** to add, edit, or remove user-defined namespaces in the message map.
  g) Select **Documentation** to provide a description of the message map, or other relevant usage notes.

What to do next

Continue editing the map, and define transformations between the input message and the output message. For more information, see "Editing message maps" on page 1394.

Adding input and output messages
Use the Graphical Data Mapping editor to add input and output messages to your message map.

Before you begin

Create a message map by using the Graphical Data Mapping editor. For more information, see "Creating a message map" on page 1382.

About this task

You can add an input object to your message map by using the **Add an input object** icon in the toolbar of the Graphical Data Mapping editor:

Note: In IBM App Connect Enterprise, you can only add one input object to a message map.

You can add an output object to your message map by using the **Add an output object** icon in the toolbar:

Note: In IBM App Connect Enterprise, you can only add one output object to a message map.

What to do next

When your message map contains all of the required input and output objects, create the connections between them, as described in "Adding connections between input and output elements" on page 1419.

Mapping xsd:any on an input or output message
You can use the **Cast** function to redefine parts of the input or output model in a message map. You can also use the **Add User-Defined** function to redefine dynamically parts of the input or output model.

In your integration solution, you can create a generic message model, which you can later redefine to a specific model, by using a wildcard, defined as `xsd:any`.

You can redefine an `xsd:any` element in any of the following ways:

- Using the **Cast** function. You can redefine an `xsd:any` element by specifying the specific complex or global type defined in a particular schema file.
- Using the **Add User-Defined** function. You can redefine an `xsd:any` element by specifying the specific complex or global type directly in the message map.
- Using a transform. You can define a transform, such as the **Submap** transform, and define the input and output `xsd:any` elements within the nested map of the transform.

Note: It is recommended to qualify an `xsd:any` element before you define any transforms in a main map. Alternatively, you can define a **Submap** between the `xsd:any` element and the output element, and then define the transforms within the nested map associated to the **Submap**.

For example, in IBM App Connect Enterprise, a SOAP message is a common example of a generic model in which you are required to define the business data being exchanged through the SOAP protocol. The predefined SOAP message format defines only the structure of the SOAP envelope and allows you to redefine the Header and Body content.

Note: The `xsd:any` input element cannot be involved in a transformation when it is contained within a cast item group. You can either create transformations on the cast elements or remove all associated cast elements to directly transform the `xsd:any`.

Qualify xsd:any parts of a schema-based message model by using the Cast function

You can define a wildcard in a schema-based message model as an `xsd:any` element to create a flexible message model that can be redefined later.

You use the **Cast** function to redefine parts of the input or output model in a message map.

For example, you might have a base type of `AddressType`, and two derived types of `USAddressType` and `CanadianAddressType`. Using the **Cast** function in the Graphical Data Mapping editor, you can cast `AddressType` to `CanadianAddressType`.

For more information, see "Casting elements in a message map" on page 1398.

Qualify xsd:any parts of a message model that does not have a schema by using the Add User-Defined function

You use the **Add User-Defined** function to redefine parts of the input or output model directly in a message map.

For more information, see "Defining user-defined elements" on page 1408.

Qualify xsd:any parts by using a Submap transform

You can use the **Submap** transform to qualify an `xsd:any` element defined on the input message assembly, the output message assembly, or both.

You must specify the input element type and the output element type using a global type in the referenced submap of the **Submap** transform.

You qualify the `xsd:any` elements in the nested map by defining the input and output elements.

Casting elements in a message map
In the Graphical Data Mapping editor, use the **Cast** function to redefine parts of the input or the output model in a message map. This function is also known as a **`mapping cast`**.

About this task

To create a flexible message model, you can define elements in your XML schema as wildcards, or you can provide an extension or derived type from a base type. In a JSON schema, you can use the "oneOf" or "anyOf" keywords to allow flexibility.

A *wildcard* is an element in your XML schema that is defined as `xsd:any`, `xsd:anyType`, or `xsd:anySimpleType`.

A message model schema might contain one or more wildcards. In addition, the model for the Message Assembly Environment, Local Environment, transport header folders, and SOAP and JSON domain messages, include wildcards, such as under the environment `Variables` folder, which allow you to define data specifically for the operation of your flow.

In the Graphical Data Mapping editor, you can redefine these wildcards by using the **Cast** function to select an element or type that is defined in a message model, such as an XML schema, DFDL schema, JSON schema, or Swagger document. For more information, see "Casting a wildcard defined as xsd:any into a specific type for a SOAP message" on page 1400 and "Configuring a generic type in the local environment tree by using the Cast function" on page 1466.

If you do not have a message model that provides a suitable element for casting, consider adding a user-defined element, as described in "Defining user-defined elements" on page 1408. When you have added a user-defined element, you can set its type from a global type in your message model, as described in "Defining the structure of a complex user-defined element" on page 1414.

When you apply a mapping cast, the choices that are available vary according to the type. The following choices are available for the specified type:

xsd:any
All global elements defined in the message models that are contained in the same application or referenced library, and are therefore visible to the map. This includes elements from XML, DFDL, and top-level objects and arrays in JSON schema.

xsd:anyType
All global types defined in the message models that are contained in the same application or referenced library, and are therefore visible to the map. This includes global types from XML, DFDL, and the types defined for top-level objects and arrays in JSON schema.

xsd:anySimpleType
All the standard XML types and any global type extensions for XML simple types defined in the XML message models that are contained in the same application or referenced library, and are therefore visible to the map.

<namespace>:<base XML schema type>
All the XML types that are defined as extensions or derived from this XML schema type.

JSON type defined with "oneOf" or "anyOf"
The JSON types defined in the `"oneOf"` or `"anyOf"` array.

If you are casting to an element or type that is derived from a JSON schema or Swagger document, see "JSON schema requirements for message maps" on page 1549.

You can use the **Cast** function to redefine a wildcard or base type multiple times. Each type that you cast appears under the wildcard or base type within a choice group.

To define transformations between subtypes of an XML data type, you can use extension or derived types. A *derived type* is a data type that is related to another data type known as the base type or super type. You can cast a base type to a derived type by using the **Cast** function. For more information, see "Casting an XML schema base type to a derived type or extension type" on page 1403.

When your input message or your output message is the IBM provided JSON domain object or JSON domain array message, you can redefine the **Data** element that is of type **anyType** by using the **Cast** function. For more information, see “Modeling a JSON message for use in a message map by using an equivalent XML schema model” on page 1572.

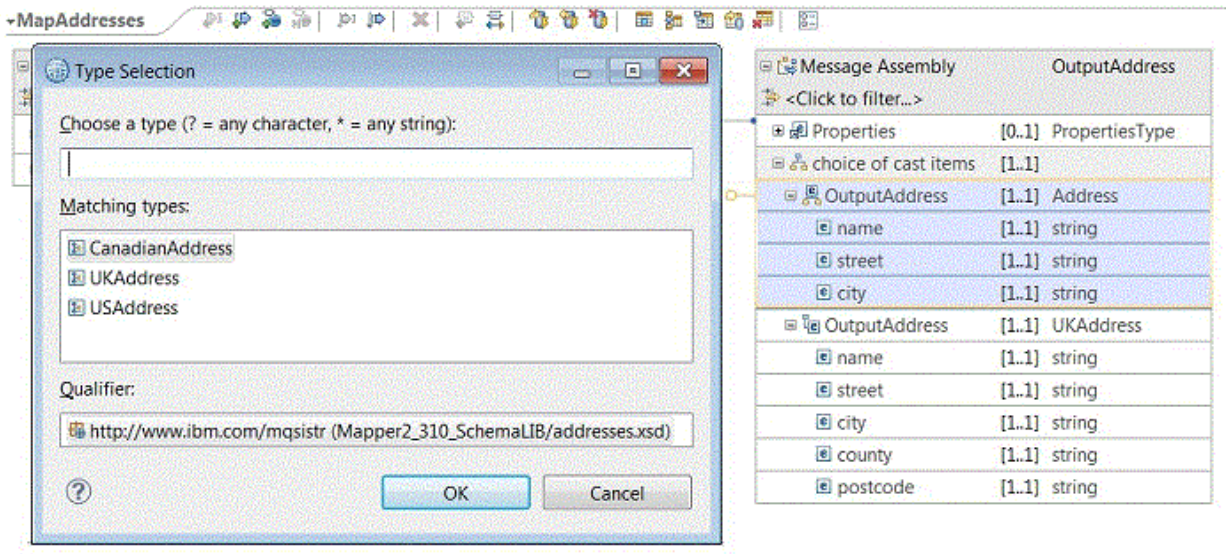
You can also use the JSON schema `"oneOf"` and `"anyOf"` keywords, to structure and combine more complex JSON schema. For more information, see “Casting with JSON schema types” on page 1405 and “JSON schema requirements for message maps” on page 1549.

Procedure

To redefine an element in your input or output model by using the **Cast** function, complete the following steps:

1. Open the map in the Graphical Data Mapping editor.
2. Right-click the element that you want to redefine, and select **Cast**.

   The **Type Selection** dialog opens:



3. Select the type that you want to cast to, and then click **OK**.

   The **Type Selection** dialog lists only those elements and types that are appropriate for the element that you want to cast, and that are contained in a referenced application or library. When you cast a base element, the **Type Selection** dialog lists only derived types.

   Note: In the following situations, **choice of cast items** contains more than one item:

   - You cast a wildcard input type to a specific schema type.
   - You cast a base schema input type to an extension or a restricted type.

   When applying XML extension or derived type schema type casts, the Graphical Data Mapping editor automatically applies an XPath expression to the mappings from these casts. At run time, the XPath expression checks the `xsi:type` attribute of the input element to determine which of the types is present in the input data. Then, only the mapping from the appropriate type is performed. The expression checks only the name of the type in each cast; the namespace is not considered. If **choice of cast items** includes types that have the same name but are in different namespaces, you must add your own conditional transform that checks the namespace and name part of the `xsi:type` attribute of the input data. When **choice of cast items** includes only a single type, the `xsi:type` attribute is not checked, and you must ensure that the input data presents only the single cast type.

Results

After you use the **Cast** function to qualify the type of the element that is defined with a wildcard, the type appears under the wildcard. You can define transformations between the input message and the output message. For more information, see "Specifying a transform (mapping operation)" on page 1430.

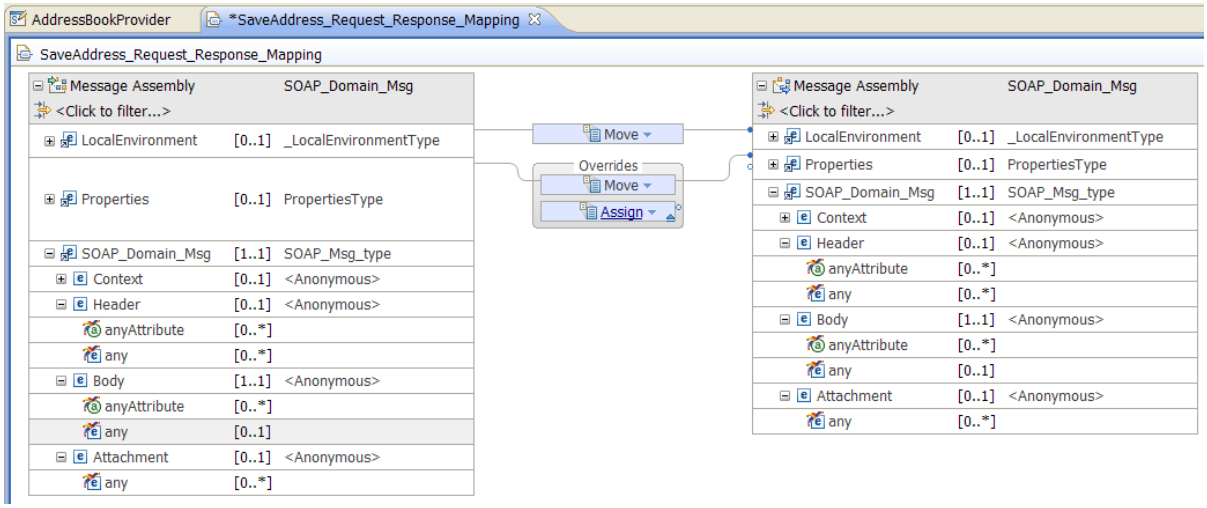
Casting a wildcard defined as xsd:any into a specific type for a SOAP message
You can use the **Cast** function to redefine a wildcard element, that is, an element with type *xsd:any*, into a specific type. Each type is described by a schema.

About this task

You can transform a SOAP message that is defined by using the predefined format **SOAP_Domain_Msg**. This message type contains a Header, a Body, and an Attachment part. Each part contains an element that is named **any** to represent a wildcard, that is, an element of type *xsd:any*. The Header and Body sections also include an element that is named **AnyAttribute**. You can cast elements and attributes included in any of these SOAP sections by using the **Cast** function.

Note: When you transform a SOAP message, you cast the Body wildcard on the input side to the type of the request message for the SOAP operation. On the output side, you cast the Body wildcard to the type of the response message for the SOAP operation.

The following figure shows the message map in the Graphical Mapping Data editor after you create a message map to transform a SOAP message:

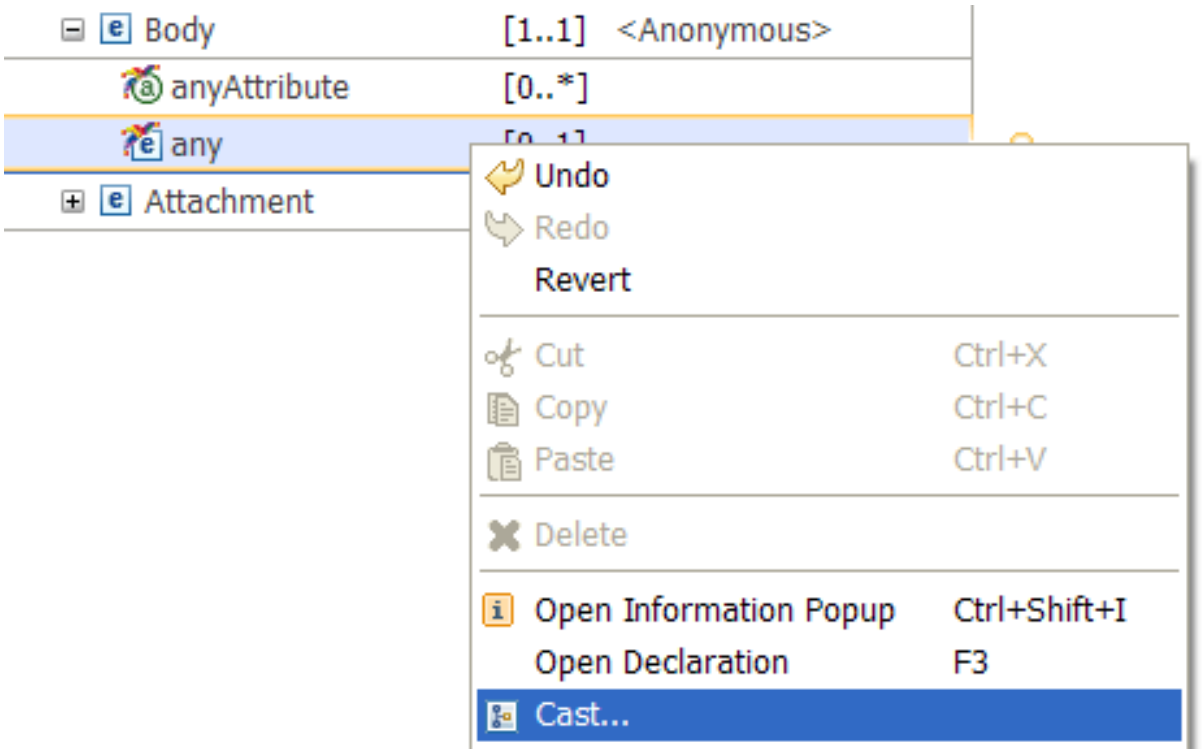


Procedure

To cast an element that is described as **any** or as **anyAttribute** in the message map, complete the following steps:

1. Right-click the element **any** or **anyAttribute** located in the section of your **SOAP_Domain_Msg** where you want to specify a type, and then select **Cast**.
   - To cast a SOAP header wildcard element, right-click **Header**, and then select **Cast**.
   - To cast a SOAP body wildcard element, right-click **Body**, and then select **Cast**.
   - To cast a SOAP attachment wildcard element, right-click **Attachment**, and then select **Cast**.

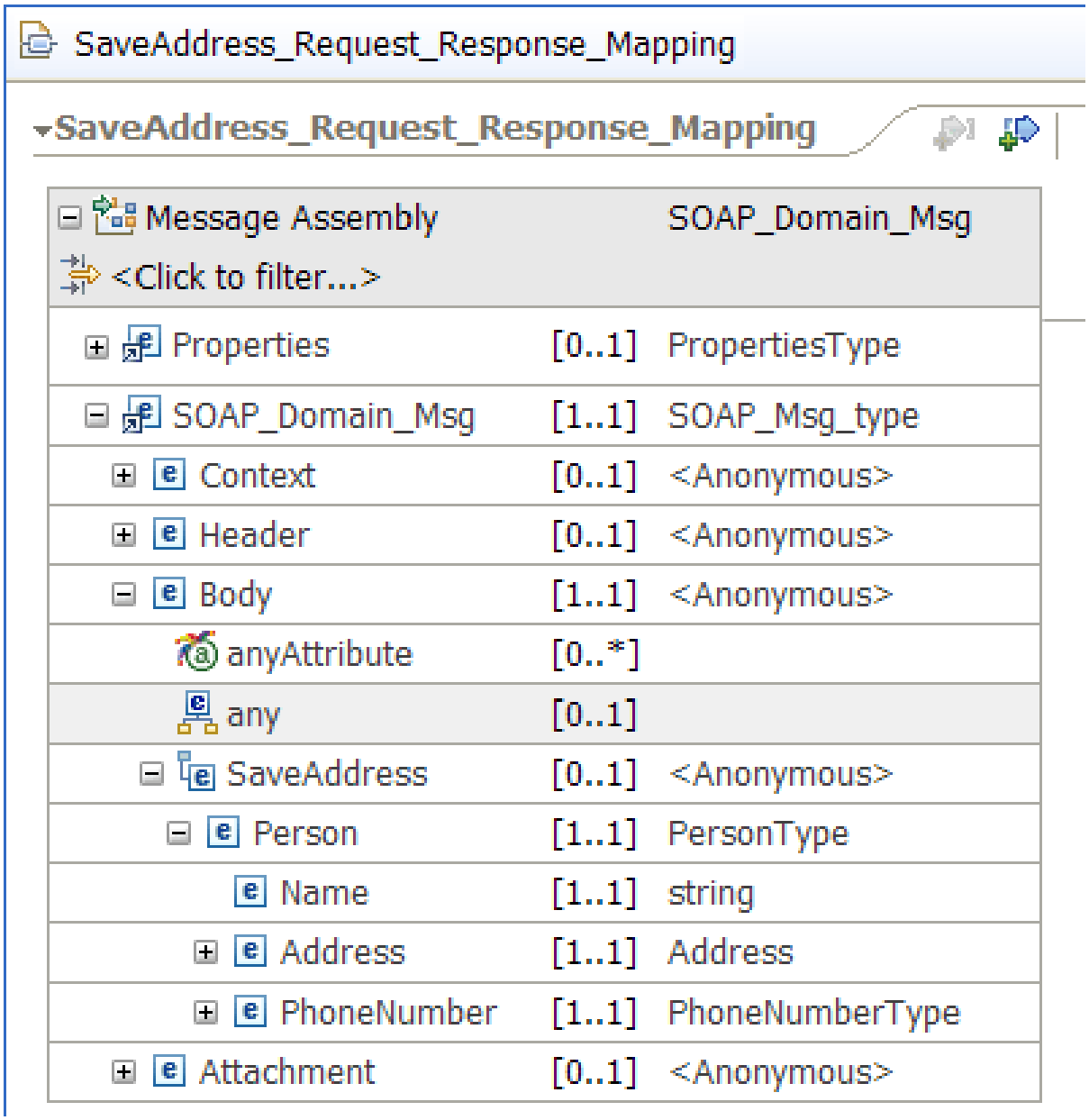
   For example, to cast the Body section, right-click **Body**, and then select **Cast**.



2. In the **Type Selection** window, select a type.

   The **Type Selection** window displays all the specific types that are available for selection. These types include the input and output elements that are defined in the WSDL file that describes your SOAP message.

   In the example, the element **any** of the **SOAP_Domain_Msg** Body is redefined to the complex element **SaveAddress**.



Results

A wildcard element is redefined to a specific type.

What to do next

Define transformations between the input message assembly and the output message assembly. For more information, see “Specifying a transform (mapping operation)” on page 1430.

Casting an XML schema base type to a derived type or extension type
In a message map, you can cast an XML schema base type to a derived type or extension type so that you can define transformations between subtypes of a data type.

Before you begin

- Model the schemas that correspond to the XML schema base type and the derived type.
- Cast an `xsd:any` element in your message map to an XML schema base type.

About this task

A *derived type* is a data type that is related to another data type known as the base type or super type.

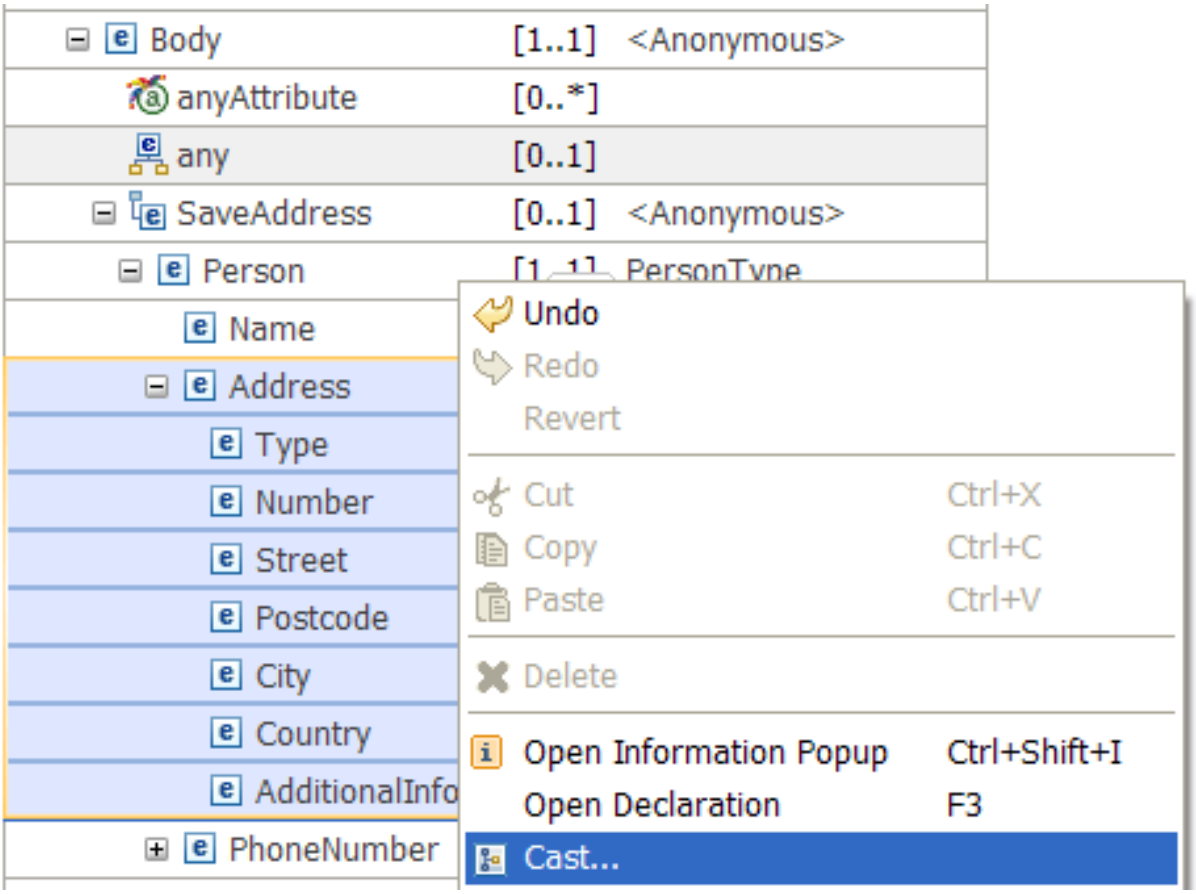
For example, **Address** is the XML schema base type, and **USAddress**, **CanadianAddress**, and **UKAddress** are derived types of **Address**.

Procedure

To cast an XML schema base type to a derived type, complete the following steps:

1. Select the XML schema base type.

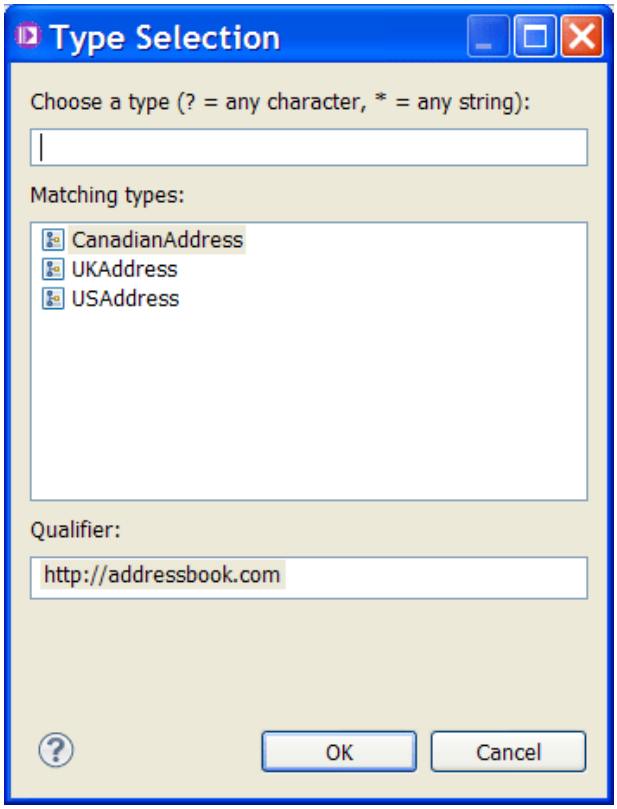
   For example, you can cast an `xsd:any` element to the **Address** XML schema base type.
2. Right-click the XML schema base type (**Address**), and then select **Cast**.



3. In the **Type Selection** window, choose a matching type, and then select **OK**.

   The options available correspond to specific address types in the schema model that are modeled by using **Address** as the XML schema base type.

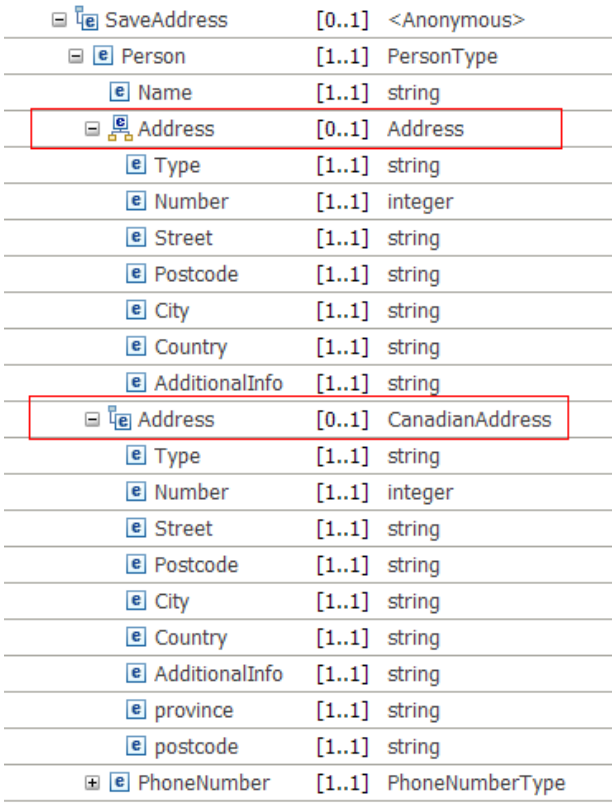
   The following figure shows the **Type Selection** window that you get:



Results

The message map contains two entries, one for the XML schema base type and a second one for the derived type.

In the example, one entry corresponds to the XML schema base type **Address**. The other entry corresponds to an Address with the derived type **CanadianAddress**.



What to do next

Define additional transformations between elements in the message map. For more information, see “Transform types in the Graphical Data Mapping editor” on page 1282.

Casting with JSON schema types
In a message map, you can use the cast function to define wildcards to a JSON data type, or extend a JSON type that is defined with the `"oneOf"` or `"anyOf"` keywords.

Before you begin

- Create a message map for your JSON data. For more information, see “Creating a message map” on page 1382.
- Ensure that you have a supported JSON schema model in the same container as your message map. For more information, see “JSON schema requirements for message maps” on page 1549.

About this task

You can use a mapping cast with JSON schema types when you want to use wildcards or combine complex JSON schema; for example:

- When your message model includes a wildcard. A wildcard is an element in your schema that is defined as `xsd:any` or `xsd:anyType`. The model for the Message Assembly Environment and Local Environment folders include wildcards (under the Variables folder, for example), which allow you to define data specifically for the operation of your flow.
- When a JSON schema includes the `"oneOf"` or `"anyOf"` keywords, to allow the flexibility to extend the content of a data type.

Procedure

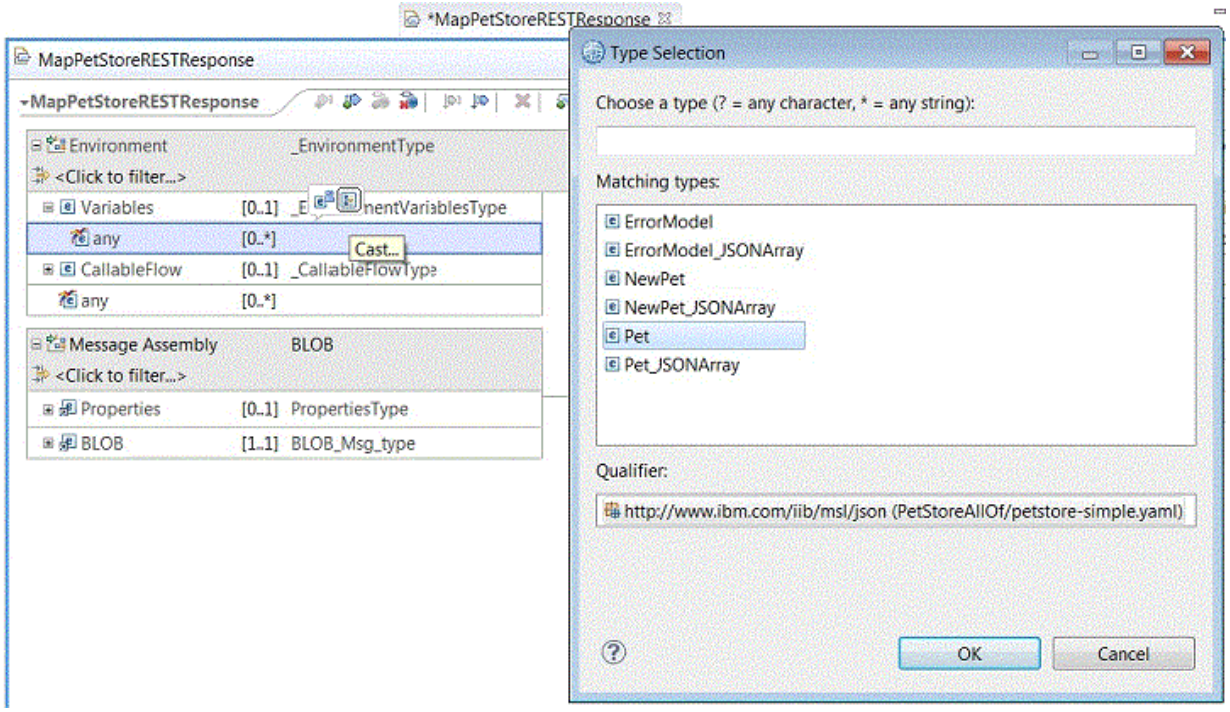
To cast a wildcard or a JSON schema type that includes the `"oneOf"` or `"anyOf"` keyword, complete the following steps:

1. Select the wildcard or JSON schema type that you want to use.
2. Right-click the wildcard or JSON schema type, and then select **Cast**.

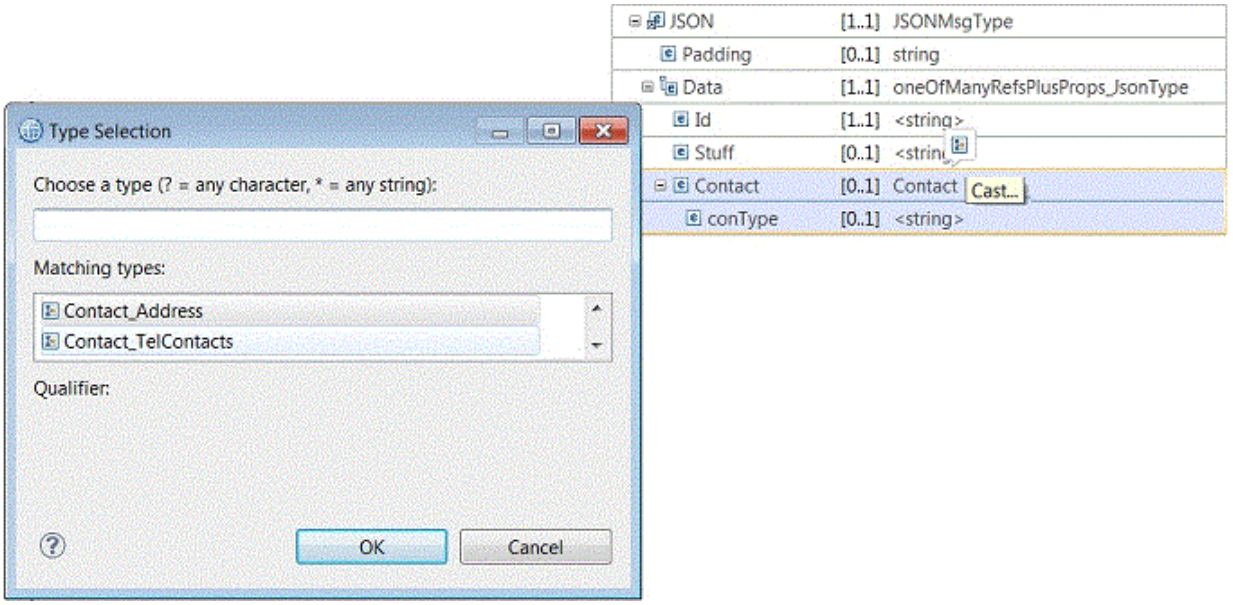
3. In the **Type Selection** window, choose the required type, and then select **OK**.

   The options depend on the JSON schema files that are available in the same project container as the message map and the item that is being cast.

   The following figure shows the **Type Selection** window that you might see when casting the environment tree `Variables` in a project container that contains a REST Swagger document used by a RESTRequest node:



The following figure shows the **Type Selection** window that you might see when casting a JSON type `Contact` that is defined using the JSON keyword `"oneOf"`, which extends the type as either an address or telephone contact:



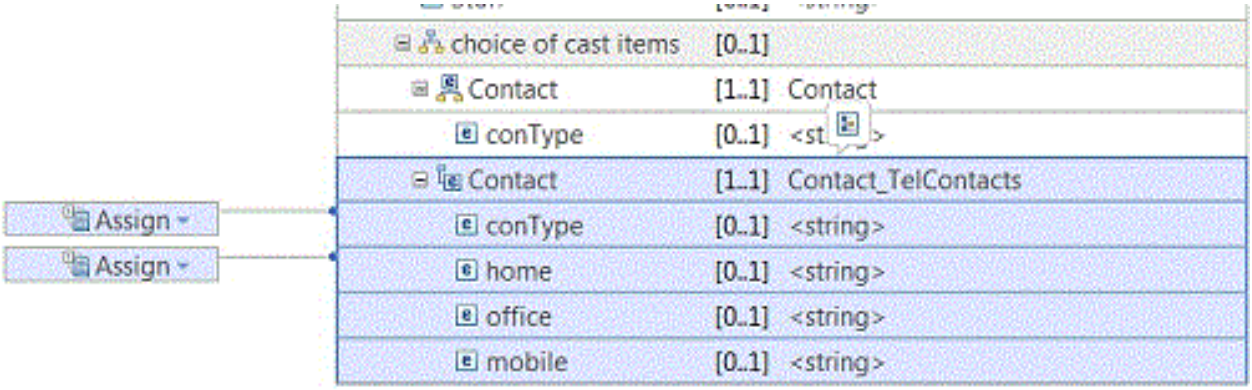
For example:

```
"Contact" : {
     "properties": {
             "conType": {
```

```
                    "type": "string"
                }
        },
         "oneOf": [
{"type": "null"},
{"$ref":"#/definitions/Address"},
{"$ref":"#/definitions/TelContacts"}
                  ]
}
```

Results

The message map contains a `choice of cast items` group, including the type that you have picked for the cast action; you can now complete mappings to the data members of that type. For example:



What to do next

Define additional transformations between elements in the message map. For more information, see “Transform types in the Graphical Data Mapping editor” on page 1282.

Defining user-defined elements
You can add, reuse, rename, transform, and delete a user-defined element directly into your message map during the development phase.

About this task

A *user-defined element* is an element that you can add directly into a message map to define the data of an extension point, which is shown as **any** or **anyAttribute** in the map.

You can add user-defined elements to define extension points in any of the following message assembly components:

- The local environment tree variables folder
- The message body
- `xsd:any` elements in your message body
- The Environment tree
- The transport headers that include extension points such as MQRFH2

You can also use user-defined elements to define a JSON message, and a SOAP or XML message that has an **xsd:any**.

You can add any of the following types of user-defined elements:

- Simple type user-defined elements
- Complex type elements
- Repeatable user-defined elements

When you add a user-defined element, you must consider the following behavior:

- You can define a simple type user-defined element or a complex user-defined element in the input message assembly or in the output message assembly.
- You can include simple type attributes, simple type elements, complex type elements, and repeatable elements in a user-defined complex type element.
- You can define user-defined attributes within a complex user-defined element.
- You can set the type of a user-defined element or attribute from an existing global type definition. You can use any available message models, XML schemas, or DFDL schemas. For more information, see "Choosing the project structure when you create a message map" on page 1380.

JSON elements are defined as nillable by default to allow the JSON null value. The Graphical Data Mapping editor creates all JSON user-defined elements as nillable.

Note: Be aware of the following limitations:

- The Graphical Data Mapping editor does not create any other type of user-defined element as nillable. For this reason, if you use the **Add User-Defined** function to add XML elements, they cannot be defined as nillable. If you need nillable XML elements, you must define a schema model for them.
- You cannot use user-defined elements as the input or output of a submap.

When you use the Graphical Data Mapping editor to transform a JSON message, ensure that the JSON element names that you define by using the **Add User-Defined** function are valid according to the XML rules for naming elements. JSON allows a wider range of characters than XML, for example a JSON object might include the **@** character. However, the Graphical Data Mapping editor allows element names that are valid in XML or XPath.

Procedure

You can do any of the following actions with user-defined elements:

- Create a user-defined element by using the **Add User-Defined** function. For more information, see "Adding and renaming a user-defined element" on page 1410.
  - You can create a simple or a complex user-defined type. For more information, see "Defining the structure of a complex user-defined element" on page 1414.
  - You can rename a user-defined element.
  - You can set the cardinality. For more information, see "Configuring the cardinality of a user-defined element" on page 1412.
- Reuse multiple times a user-defined element. For more information, see "Reusing a user-defined element in a message map" on page 1415.
- Specify transforms where a user-defined element can be an input element to the transform, or an output element. For more information, see "Specifying a transform (mapping operation)" on page 1430.
- Delete a user-defined element. For more information, see "Deleting a user-defined element" on page 1417.

What to do next

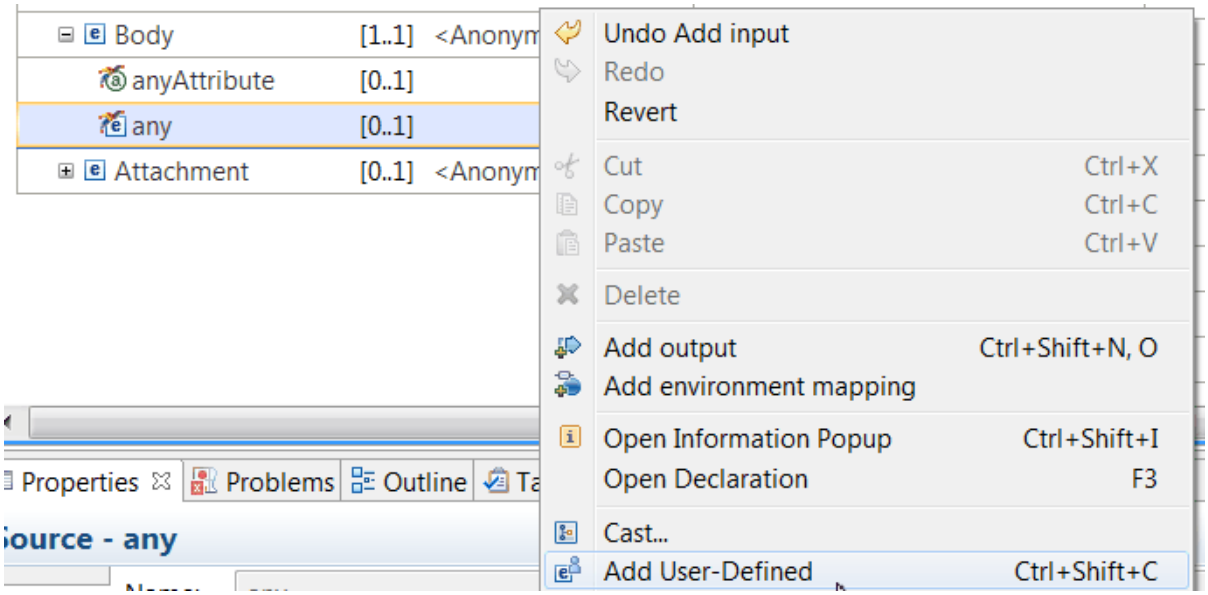
After you add your input and output objects, create the connections between them, and specify your transforms, you can test your message map. For more information, see "Troubleshooting a message map" on page 1587.

Adding and renaming a user-defined element
You can add a user-defined element by using the **Add User-Defined** function. You can add simple type or complex type user-defined elements.

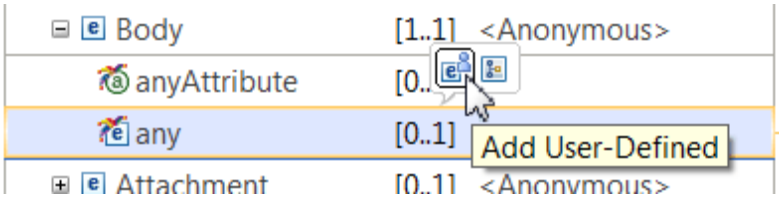
Procedure

To create a user-defined element, complete the following steps:

1. Select the extension point **any** under which you want to add a user-defined element.
2. Select **Add User-Defined**. You can choose any of the following methods:
   - Right-click the element any. A menu appears. Select the **Add User-Defined** function.



   - Select the **Add User-Defined** function by selecting the **Add User-Defined** icon  that is available on top of the field.



   An element is added under the **any** field.

   The default type of a new created user-defined element is **string**.
3. Click the user-defined element, and then enter the name of the element.

   If the user-defined element name exists in multiple places, you are prompted to select an action that determines whether to reuse or rename an existing user-defined element in multiple places in the map.

4. Specify the user-defined element type. Complete the following steps:
   a) Click the type of the user-defined element.
   b) Select the type of the user-defined element.

      Note: The default type is **string**.

      - To configure a simple type, choose from any of the following values:
        - Boolean
        - date
        - dateTime
        - decimal
        - double
        - duration
        - float
        - hexBinary
        - int
        - long
        - string
        - time
      - To configure a complex type, choose **<Anonymous>**.
      - To configure a JSON array, choose **JSONArray_Default**.
      - To set the type by using a global XML schema, choose **Browse**, and select the type. You can set the type to an XML schema or to a DFDL schema.

        When you define the user-defined element type by using a global XML schema, the following rules apply:

        - You can have the map and any schema definitions that are used in the map in the same project. The project can be a static library project, a shared library project, or an application project.
        - You can have any schema definition that is used in the map in a static library project and the map in an application project.
        - You can have any schema definition that is used in the map in a shared library project and the map in a different shared library project.
        - You cannot have the schema definitions that are used in the map in a shared library and the map in an application project. The Graphical Data Mapping editor reports error **CWMSL209E** if you break this rule.

        Note: You can get error **CWMSL209E** when the type of the user-defined element references an unresolved schema. To resolve this problem, add the schema in the Scope tab of the map **Properties** view, or add the dependency to the map container project. Then, reselect the type.
5. When you add elements to a complex user-defined element, you define the cardinality for each child element that you add. The cardinality defaults to **Minimum occurrence** and **Maximum occurrence** of 1. For more information, see "Configuring the cardinality of a user-defined element" on page 1412.

   Note: You cannot change the cardinality for user-defined elements that are within the choice group that is created for you when you define elements at the level of the **xsd:any** wildcard. The cardinality is determined by the cardinality set on the **xsd:any**.

What to do next

- If you add a simple type user-defined element, edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see "Specifying a transform (mapping operation)" on page 1430.

- If you add a complex type user-defined element, define its complex structure by using the **Add User-Defined** function. For more information, see "Defining the structure of a complex user-defined element" on page 1414.
- If you add a JSON array, define its structure by using the **Add User-Defined** function. For more information, see "Graphically modeling a JSON array in a message map" on page 1560.

Configuring the cardinality of a user-defined element
In a message map, you can set the cardinality of user-defined elements except for JSON arrays and elements that are defined at the **xsd:any** level.

About this task

In the Graphical Data Mapping editor, when you define your message data with user-defined elements, you must comply with the following rules:

- The Graphical Data Mapping editor sets the cardinality for elements that are defined at the **xsd:any** level based on the internal model of the **xsd:any**.
- You can configure the cardinality of user-defined elements in the **Properties** page of the element.
- You cannot set the cardinality of elements that are defined at the **xsd:any** level because their value is defined by the cardinality set in the message model that provided the **xsd:any**. Typically, models define the **xsd:any** cardinality as **Minimum occurrence** = `0` and **Maximum occurrence** = unbounded so that user-defined elements defined at the**xsd:any** level can optionally exist, and optionally repeat. The predefined JSON and SOAP message models use this approach.
- You can change the cardinality of a user-defined element of type **<Anonymous>** that is not defined at the **xsd:any** level.
- You can change the cardinality of any child user-defined element that describes a user-defined element of type **<Anonymous>**.
- If you set the type of a simple user-defined element with an external message model, you cannot change the cardinality of the element in the **Properties** page. The cardinality of this element is set from the schema model that you reference.
- If you set the type of a complex user-defined element with an external message model, you cannot change the cardinality of its children in the **Property** pages. The cardinality of these elements is set from the schema model that you reference.
- In a JSON message, the following rules apply for JSON arrays:
  - You cannot change the cardinality of a user-defined element of type **JSON array** that defines a JSON array in the root of a JSON array message.
  - You cannot change the cardinality of **Item** in a JSON array.
  - If you change the type of **Item** to **<Anonymous>**, you can change the cardinality of any child elements, including JSON array elements.

The cardinality represents the number of occurrences of an element. You can define an element as mandatory, optional, or repeating. You set the cardinality of an element by configuring the **Minimum occurrence** and the **Maximum occurrence** properties.

- An *optional* element has **Minimum occurrence** set to `0`. An optional element can be omitted from the message.
- A *repeating* element has **Maximum occurrence**> 1 to indicate a bounded number of repeats, or **Maximum occurrence**=unbounded to indicate an unlimited number of repeats. A repeating element occurs more than once in the message, and all the occurrences must appear together without any other elements between them.

Procedure

In the **Properties** page of a user-defined element, complete the following steps to set its cardinality if the element is not defined at the **xsd:any** level, and the element is a child of a user-defined element:

- Check whether the default cardinality of a user-defined element meets your message model requirements or not.

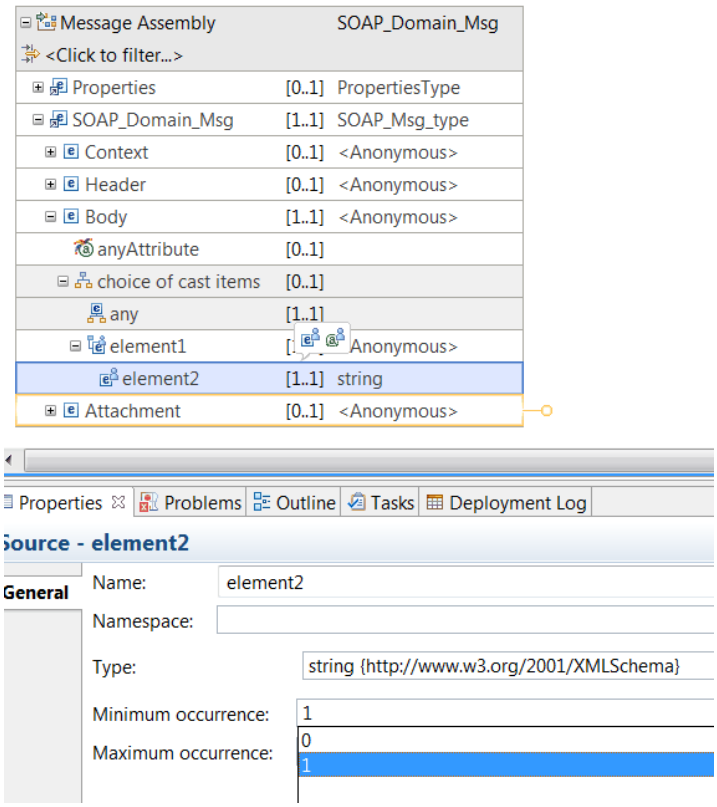
  The Graphical Data Mapping editor sets the cardinality of user-defined elements to **1:1**, that is, any user-defined element must be present exactly once.

  - **Maximum occurrence** = 1
  - **Minimum occurrence** = 1
- Set the cardinality of any user-defined element that you add below the **xsd:any** level if your data does not match the default configuration.
  - Set the **Minimum occurrence** to 0 or 1.
  - Set the **Maximum occurrence** to 0, 1, or unbounded.

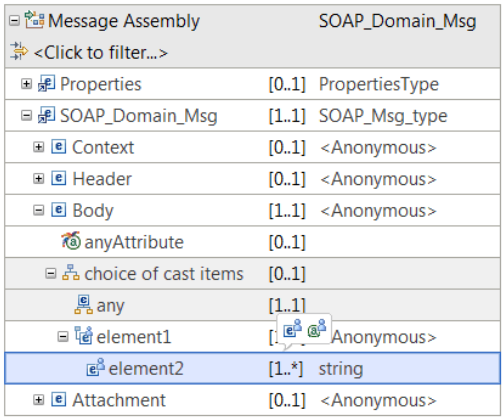
Example

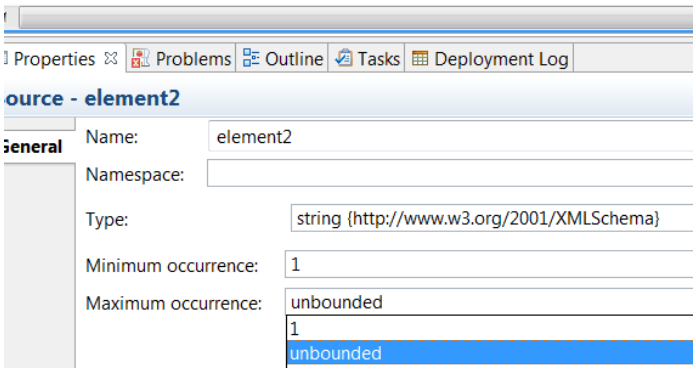
When you have a simple or complex user-defined element that is not defined at the **xsd:any** level, you can configure the cardinality of the element. The following figure shows an input SOAP message with a user-defined element named *element2*. You create this element by using the **Add User-Defined** function.

To change the default value of **Minimum occurrence**, open the **Properties** page of the element, and select 0 or 1.



To change the default value of **Maximum occurrence**, open the **Properties** page of the element, and select 0, 1, unbounded, or enter a number as required.





What to do next

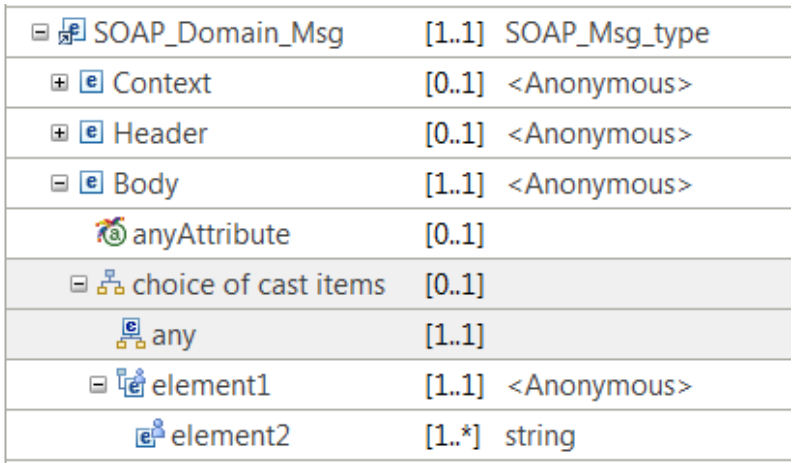
Construct the transforms to map the data you model. For more information, see "Choosing a transform to map repeating elements" on page 1362 or "Specifying a transform (mapping operation)" on page 1430.

Defining the structure of a complex user-defined element

You can define the structure of a complex type by using the **Add Element** function. You can reorder the position of elements in your complex structure. You can also use global types such as XML schemas, DFDL schemas, JSON schemas, and Swagger documents to set the type of user-defined elements.

Before you begin

In a message map, add a user-defined element with **<Anonymous>** type to your input or output assembly. For more information, see "Adding and renaming a user-defined element" on page 1410.



About this task

You can define a complex user-defined element in the input message assembly or in the output message assembly.

- You can include simple type attributes, simple type elements, complex type elements, and repeatable elements in a user-defined complex type element.
- You can set the type of a user-defined element to a global complex type defined in an XML schema, DFDL schema, JSON schema, or Swagger document.

To set the cardinality of a complex user-defined type, you must differentiate whether the type is based on a global type or not.

- When you set the type to a global type, you can configure the cardinality only through an XML schema or DFDL schema.
- When you add elements to a complex user-defined element, you define the cardinality for each child element that you add. The cardinality defaults to **Minimum occurrence** and **Maximum occurrence** of 1. For more information, see "Configuring the cardinality of a user-defined element" on page 1412.

  Note: You cannot change the cardinality in the Graphical Data Mapping editor for user-defined elements that are within the choice group that is created for you when you define elements at the level of the **xsd:any** wildcard. The cardinality is determined by the cardinality set on the **xsd:any**.

Procedure

Complete the following steps to define the structure of a complex element:

- Right-click the complex user-defined element, and select **Add Child Element** from the context menu.

  You can define simple type elements and complex type elements.
- Specify the type of the element. Click the type of the element and choose one of the XML simple types, or select **Browse** to set the type from a global complex type from an XML schema, a DFDL schema, or a JSON schema or Swagger document that is in the same project container as the map. If you are casting to a type that is derived from a JSON schema or Swagger document, see "JSON schema requirements for message maps" on page 1549 for more information.
- Modify the location of an element in your structure. Right-click the element that you want to move, and choose **Move down** to move down an element or **Move up** to move up an element.
- Specify the cardinality of the element. Configure in the element **Properties** tab a value for the properties **Minimum occurrences** and **Maximum occurrences**. For more information, see "Configuring the cardinality of a user-defined element" on page 1412.
- Repeat the previous steps to define all the elements.

What to do next

Edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Reusing a user-defined element in a message map

You can reuse multiple times a user-defined element that you have modeled in the input message assembly or in the output message assembly. All instances of the reused user-defined element must be of the same type.

About this task

You can reuse user-defined elements in the same message map or in a different message map.

To reuse a user-defined element in multiple maps, you can define the user-defined element in a submap. Alternatively, you can copy and paste the user-defined element between maps. When you copy a user-

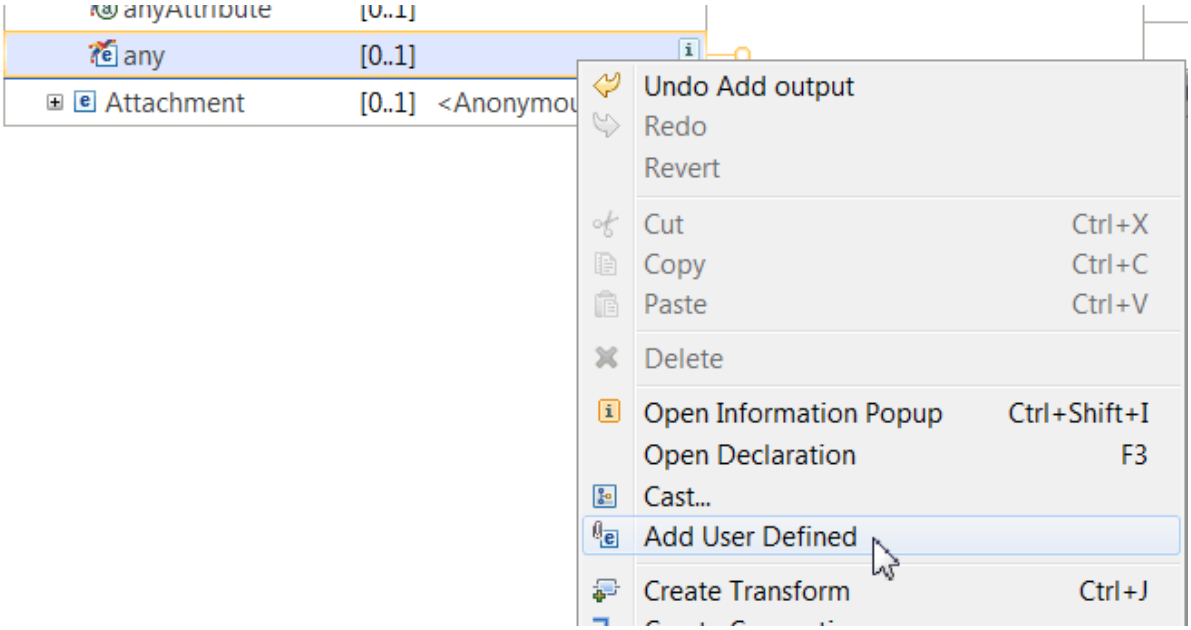
defined element, you must maintain in multiple places any changes you make to the user-defined element.

Procedure

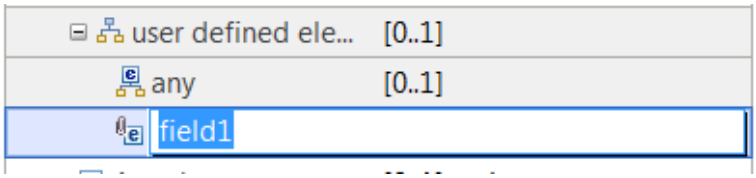
Complete the following steps to copy into the output message assembly a user-defined element that is defined in the input message assembly:

1. Add a user-defined element to your input message assembly. For more information, see "Adding and renaming a user-defined element" on page 1410.

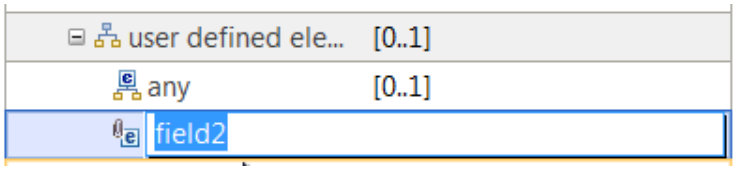
   You can select the **Add User-Defined** function to add an element.



   A user-defined element is added. In this example, the user-defined element name is `field1`.



2. Add another user-defined element. In this example, you add a user-defined element on the output message assembly. The element name is `field2`, which is the next available generic name for user-defined elements.



3. Rename `field2` to **field1**.

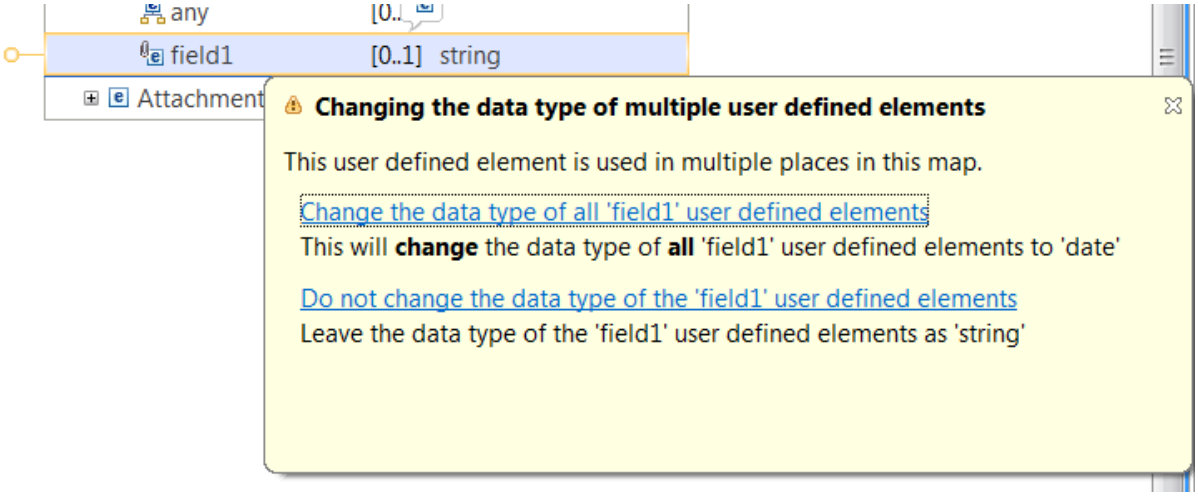
   If the user-defined element name exists, you are prompted to select an action that determines whether to reuse or rename an existing user-defined element in multiple places in the map. You can choose any of the following actions:

   - Select `Change the name of all element1 user defined elements.` to change the name of all user-defined elements with that name in the map. The element *element1* is the name of the existing user-defined element.
   - Select `Change the name of only this element1 user defined element.` to change the name of only this user-defined element. The element *element1* is the name of the existing user-defined element.
   - Select `Do not change the name of this element1 user defined element.` to leave the name unchanged.

4. Optional: When you change the type of an element that is reused multiple times, you must specify whether all elements with the same name are updated to the new type. You can choose from any of the following options:

   - Change the data type of all the user-defined elements with the same name to a new type.
   - Do not change the data type for all the user-defined elements with the same name.

     Note: If you want to change the type of only one instance, you must change the name of the user-defined element and define it with the new type. You cannot have a user-defined element with two distinct types.



What to do next

Edit the message map, and define transformations between the input message assembly and the output message assembly. For more information, see “Mapping input to output elements manually” on page 1418.

Deleting a user-defined element
You can delete a user-defined element by using the **Delete** function from the menu options. Alternatively, you can use the **Delete** key or the **Delete** icon in the Graphical Data Mapping editor button bar.

About this task

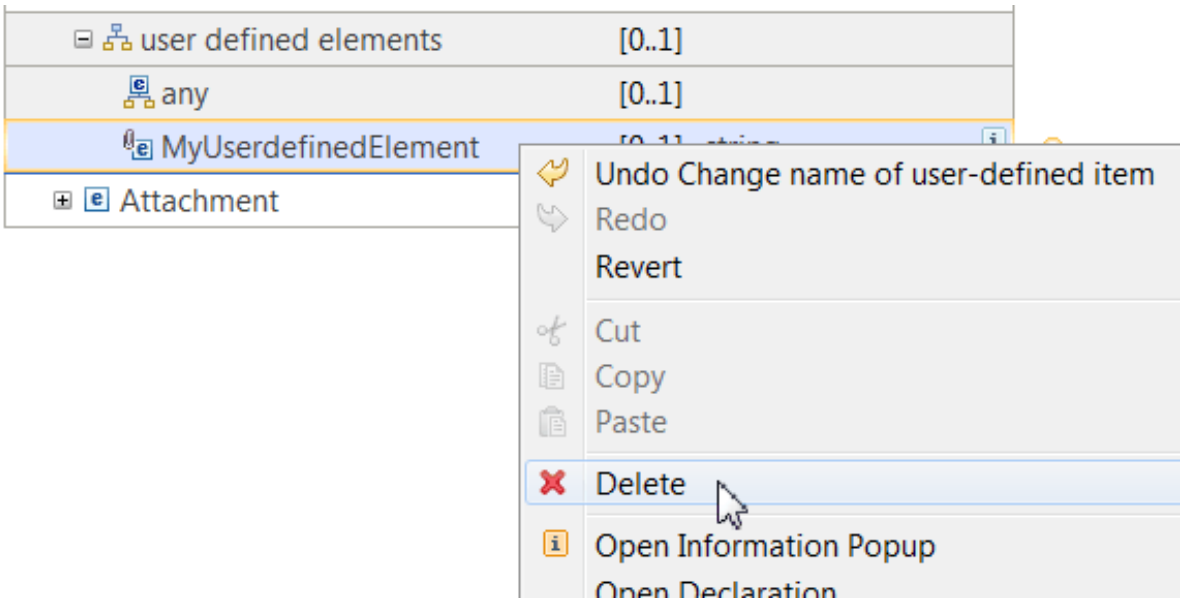
You can reuse a user-defined element in multiple locations in your input message assembly or your output message assembly if they are of the same type. However, you can only delete one user-defined element entry, that is the entry that you select for deletion. The rest of instances where the user-defined element is defined remain intact.

Procedure

Complete the following steps to delete a user-defined element:

1. Identify the user-defined element that you want to delete in the input message assembly or the output message assembly.
2. Right-click the element.

   A menu appears.



3. Select the **Delete** function and click **OK**.

What to do next

Continue editing the message map, and define transformations between the input message assembly and the output message assembly. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Mapping input to output elements manually
Use the Graphical Data Mapping editor to map from input to output elements.

Before you begin

- Create a message map by using the Graphical Data Mapping editor. For more information, see "Creating a message map" on page 1382.
- Add input and output objects to a message map. For more information, see "Adding input and output messages" on page 1396.

About this task

Note: You can also use the Graphical Data Mapping editor to map automatically from input to output elements. For more information, see "Mapping input to output elements automatically" on page 1421.

Procedure

To create manually the mappings between elements in a message map, complete the following tasks:

1. Add connections between the input and output objects. For more information, see "Adding connections between input and output elements" on page 1419.
2. Optional: Connect multiple input elements to a transform. For more information, see "Connecting multiple input elements to a transform" on page 1420.

3. Create the transforms. If the transform that you have selected is a type that is part of a nested transform (for example, Local map, Join, If), you must enter into that nested map and complete the transformation in it. For more information, see "Specifying a transform (mapping operation)" on page 1430.
4. Set the properties for the transforms (for example, Condition, Cardinality, or Order).

What to do next

When you have added your input and output objects, created the connections between them, and specified your transforms, you can test your message map. For more information, see "Troubleshooting a message map" on page 1587.

Adding connections between input and output elements
Use the Graphical Data Mapping editor to create connections between the input and output objects in your message map.

Before you begin

- Read the following concept topics:
  - "Graphical Mapping overview" on page 1264
- Create a message map by using the Graphical Data Mapping editor. For more information, see "Creating a message map" on page 1382.
- Add input and output objects to your message map. For more information, see "Adding input and output messages" on page 1396.

Procedure

You can create a connection in a message map by using one of the following methods:

- Move the cursor anywhere over the required element in the input object, then click and drag the connection over the required output element and drop. You can also create the connection from the opposite direction, by dragging the connection from the output element to the input element. You can also use the grab handle that appears when you hover over an element to click and drag the connection.

  Note: When you drag and drop a connection from an input element to an output element, a transform is added automatically. To add the transform, you must drop the connection above the lowest point of the input and output message. If you use this method in a nested map, you must drop the connection above the lowest point of the shaded area.

  you must drop above the bottom of the shaded area"
- Select the appropriate input or output object and right-click to display the menu, then select **Quick Link**. An outline view of the opposite data structure is displayed; if the **Quick Link** action is invoked on an input object, a quick outline view of the output data structure is shown, and if the action is invoked on an output object, a quick outline view of the input data structure is shown. You can then use the quick outline view and its built-in filter to find and select the required element. When you have selected the required element, a transform is created in the Graphical Data Mapping editor.
- Select the required input element and right-click to display the menu, then select **Add Connection**. This method allows you to carry out other actions in the editor that require the use of mouse clicks (such as expanding output elements, or using the scroll bars) while a transform is being created.
- Some transformations, such as the **Assign** transform, require a connection to an output object, but not to an input object. You can create these transformations either by invoking the **Add Connection** action, or by clicking anywhere on the output object or on its **grab handle** icon and then dragging the connection onto the empty space between the input and output objects.

What to do next

When you have created the connection between your input and output objects, select the required transform as described in "Specifying a transform (mapping operation)" on page 1430.

Connecting multiple input elements to a transform
To make additional input data available to a transform, you can add additional connections. You must define the connection type as either **Primary** or **Supplement**. A **Primary** connection controls the type and outcome of the transform. A **Supplement** connection does not change the type of the transform; it can be used to provide additional input to the transform, or in an optional condition on the transform.

Before you begin

- Read the following concept topics:
  - "Graphical Mapping overview" on page 1264
- Create a message map by using the Graphical Data Mapping editor. For more information, see "Creating a message map" on page 1382.
- Add input and output objects to your message map. For more information, see "Adding input and output messages" on page 1396.

About this task

When you create additional input connections to a transform, you must choose the connection type.

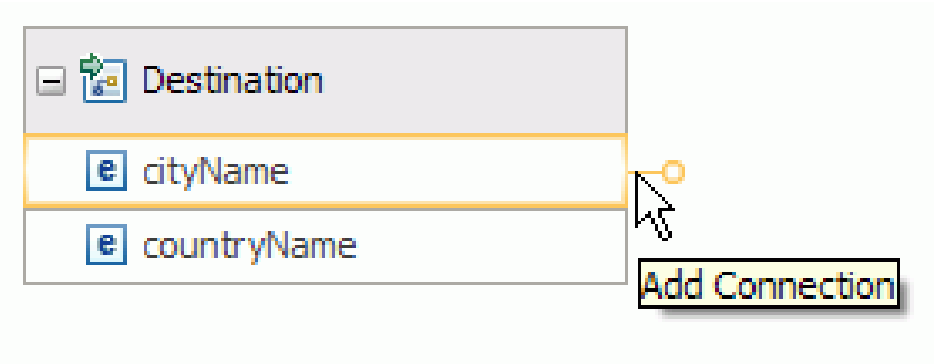
- *Primary connection*: You define a **Primary** connection when you need to control the type and outcome of the transform..
- *Supplement connection*: You define a **Supplement** connection to provide additional input to the transform, or to be used in an optional condition on the transform.

You can also change the preferences of the Graphical Data Mapping editor. You can set the **Let optional connection be primary for transforms that do not accept anymore primary connections** preference.

Procedure

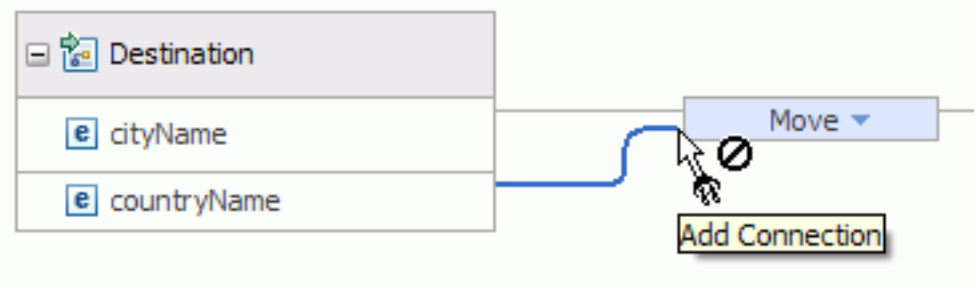
The following steps show how to connect multiple input elements to a transform by using the drag and drop method:

1. Move your cursor to the element of the input object that you want to map:



2. Click anywhere in the input element, or on its **grab handle** icon , and drag the connection to the output element. A connection is created between the two elements, and a transform is assigned, based on the number and type of input elements.

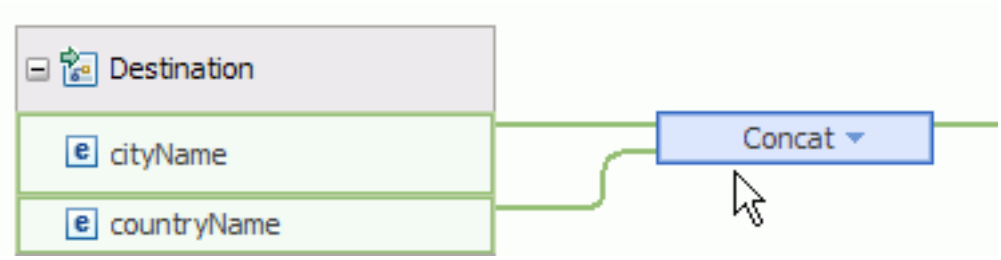
3. Create a new connection between a second input element and the transform.



When you define an additional connection between a second input element to an existing transform in your message map, you are prompted to specify if the connection is a primary connection or if it is a secondary connection (also known as a supplement connection).

4. Specify the type of the new connection as either a primary connection or a secondary connection (Supplement). If you specify a primary connection, the default transform changes to one that allows multiple primary inputs, such as Local Map, Concat, or Join.

For example:



If you choose **supplement connection**, the transform type is not changed and the additional connection is available to use within the transform. For example, you could use its value in a condition expression, or for nested transforms, such as **For each**, **Join**, or **Local map**. The additional input now appears as an extra source on the left side within the nested mapping.

What to do next

When you have created the connection between your input and output objects, select the required transform as described in "Specifying a transform (mapping operation)" on page 1430.

Mapping input to output elements automatically
Use the Auto map wizard (automap) to automatically create **Move** transforms from input to output elements based on some correlation of the names of input and output elements to create mappings.

About this task

You can use Auto map as a quick fix when you create a **Local map** or another nested mapping, and a warning or error marker is displayed to prompt you to complete the nested mapping.

Procedure

The following steps describe how to map from input to output elements by using the Auto map wizard:

1. In the Graphical Data Mapping editor, select the input and output elements that you want to map.
   - If you make no selections, the entire input and output objects in the current mapping level are processed by the Auto map wizard.
   - You can select groups of simple elements or complex elements. For complex elements, the descendants of the selected complex element can be mapped.
2. Click the Auto map icon.

   The Auto map wizard opens.
3. Choose the appropriate **Mapping scope** option:
   - **Map all simple descendants of the selected elements**. This option maps the descendants of the input element to the descendants of the output element that match each other; this option is selected by default. Optionally, select **Group transforms into nested maps**. For more information about using nested maps, see "Grouping transforms into nested maps" on page 1441.
   - **Map the immediate children of the selected elements**. This option maps only the immediate child elements of the input element to the immediate child elements of the output element that match each other.
4. Specify how names are matched in the **Name Matching Options** section.
   a) Select the **Case sensitive** option to set whether you want to match the case sensitivity of the name. This option is not selected by default.
   b) Select the **Alphanumeric characters (Letters and digits only)** option to exclude special characters (for example & and -) from the name. This option is selected by default.

   These two options are independent of each other, and you can select their values separately.
5. In the **Mapping Criteria** section, specify the mapping criteria for the matches between the input and output element names:
   - **Create transforms when the names of inputs and outputs are the same**. This option matches items of the same name, and is selected by default.

     Whether the two names are considered to be the same, depends on your selections for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**. For example, if you use the default options for **Case sensitive** and **Alphanumeric characters (Letters and digits only)**, GIVEN_NAME and GivenName are considered to be a match. However, if you select **Case sensitive** and **Alphanumeric characters (Letters and digits only)**, the two names are considered to be identical only if they contain the same alphanumeric characters in the same order, and the characters are of the same case.

     For more information, see "Mapping by same name" on page 1423.
   - **Create transforms when the names of inputs and outputs are more similar than**. With this option, you can specify how similar two names must be to create a mapping between them by varying the result from zero to 100 percent. The result is displayed and the default value is 60; see "Examples of similarity values" on page 1425 for some examples of how similar words are matched to one another.

     For more information, see "Mapping by similar name" on page 1424.
   - **Create transforms when the input and output names are matched to synonyms defined in a file**. With this option, you can create mappings for word pairs that are defined in a synonym file. A synonym file is a flat text file with file extension `.txt` or `.csv`. For further information about

creating a synonym file from a Microsoft Excel spreadsheet, see “Creating and using a synonym file” on page 1428.

For more information about the synonym file, see “Format of the synonym file” on page 1425. For information about the methods that are used to match synonyms in a synonym file, see “Algorithm used to match synonyms” on page 1428.

If the input and output names that you are using satisfy more than one of the following options, the order in which names are matched is:

a. Same

b. Synonym

c. Similar

Any output that is matched during an earlier step is excluded from name matching in a later step.

6. Click **Finish** to start the auto mapping, or click **Next** to view a summary of the transforms that were found and to choose the transforms that you want to create. You can clear matches that are defined on the **summary** page.

   Sometimes an input might match to more than one output; use the summary page to review and choose which mappings to create.

   The auto mapping is performed, and the selected matched input and output elements are mapped through **Move** transforms. Each auto mapped **Move** transform is annotated to indicate that it was created as a result of auto mapping. Hover the mouse or right-click the annotated transform to accept or reject the automatically created transforms.

Mapping by same name

Learn about the rules that apply when you select the **Create transforms when the names of inputs and outputs are the same** option in Auto map (automap).

About this task

When you select **Create transforms when the names of inputs and outputs are the same**, the following rules apply:

1. Any output field that has a fixed value is excluded in name matching. Any output that is already mapped, or under a container that is already mapped, is excluded from name matching.
2. If an input and an output have the same name, it is a match, regardless of the kind of, and XSD type of, the input and output. An element, a database column, or an attribute can all form a match if their names are the same.
3. XML namespaces are excluded from name matching. Therefore, `abc:something` and `xyz:something` are considered the same, as are `{http://www.abc.com}:something` and `{http://www.xyz.com}:something`.
4. When multiple inputs have the `same` name as one output, one mapping is created.

   However, if you have multiple inputs with the same name as one output and you choose to map by the same name (or similar name) **and** to match descendants, an attempt is made first to match by path and name. If a match is found, one transform is made, and no further matches are considered.
5. When a single input has the same name as multiple outputs, multiple mappings are created, each for one input and one output.

   However, if you have a single input with the same name as multiple outputs and you choose to map by the same name (or similar name) **and** to match descendants, an attempt is made first to match by path and name. If a match is found, one transform is made, and no further matches are considered.

6. When you select the **Map all simple descendants of the selected elements** option, the following steps are taken to match names:

   a. Compare the relative path and item name of the selected input and output

   b. Compare the item name without relative path

   For example, assume you have the following input and output items:

   - **Input:**

   ```
   OldPurchaseOrder
     items
      item
       partNum
     partNum
   ```

   - **Output:**

   ```
   NewPurchaseOrder
     items
      item
       partNum
     resource
      partNum
   ```

   If you select **Create transforms when the names of inputs and outputs are the same** when you have the inputs and outputs shown above, the relative paths of all the items are:

   - Relative paths of the input items:

   ```
   items/item/partNum
   partNum
   ```

   - Relative paths of the output items:

   ```
   items/item/partNum
   resources/partNum
   ```

   During step a) `items/item/partNum` and `items/item/partNum` are matched.

   During step b) `partNum` and `resources/partNum` are matched.

   Inputs and outputs matched in a previous step are ignored in later steps.

   When you select the **Map the immediate children of the selected elements** option, the only step taken to match names is to compare the item name without the relative path.

Mapping by similar name
Learn about the rules that apply when you select the **Create transforms when the names of inputs and outputs are more similar than** option in Auto map (automap).

About this task

When you select **Create transforms when the names of inputs and outputs are more similar than**, the following rules apply:

1. Fixed value outputs and mapped outputs are excluded in name matching.
2. The *similarity* test is done using the name of an element, a database column, or an attribute regardless of its type. If an input and an output have the same name, it is a match, regardless of the kind of, and XSD type of, the input and output. An element, a database column, or an attribute can all form a match if their names are the same.
3. The similarity test applies in the same way to case sensitivity and alphanumeric characters as for **Mapping by same name**.
4. Namespace or namespace prefixes do not participate in the similarity test. XML namespaces are excluded from name matching. Therefore, `abc:something` and `xyz:something` are considered the same, as are `{http://www.abc.com}:something` and `{http://www.xyz.com}:something`.

5. When multiple inputs have the same name as one output, one mapping is created. However, if you have multiple inputs with the same name as one output and you choose to map by the same name (or similar name) **and** to match descendants, an attempt is made first to match by path and name. If a match is found, one transform is made, and no further matches are considered.
6. When you select **Map all simple descendants of the selected elements**, the following steps are taken to match names.

   Inputs and outputs matched in a previous step are ignored in later steps:

   a. Compare the relative path and item name of the selected input and output
   b. Compare the item names without relative path
   c. Compare similar item names without relative path

   When you select the **Map the immediate children of the selected elements** option, the only step taken to match names is to compare similar item names without the relative path.
7. You can select the similarity threshold for two words to be considered similar.
8. You cannot use any other similarity algorithm.

Examples of similarity values
The following table lists words that are similar to one another, together with their similarity value, as a percentage:

Word1	**Word2**	**Similarity value %**
catalog	catalogue	85
anasthesia	anaesthesia	84
recognize	recognise	75
color	colour	66
theater	theatre	66
tire	tyre	33
intro	introduction	53
abbr	abbreviation	42
name	fullname	60
firstname	fullname	40
id	identification	14
NCName	Non colonized name	40
USA	United States of America	0
faq	frequently asked questions	0

Format of the synonym file
The Auto map (automap) facility allows you to create mappings between specific inputs and outputs by putting the names of the inputs and outputs in a file called the synonym file.

Synonyms, in the context of the synonym file, are groups of words that represent mappings that you want to create.

File type

A synonym file can reside anywhere in your file system, only if the encoding used in the synonym file is the same as that used by the Eclipse Toolkit system.

However, if the synonym file uses a specific encoding that is, or might be, different from the encoding of the Eclipse Toolkit, the file must reside in a project in the IBM App Connect Enterprise Toolkit.

If the synonym file is created outside the IBM App Connect Enterprise Toolkit, and uses a specific encoding, save the file under an IBM App Connect Enterprise Toolkit project and click **Refresh** to make the file visible in the navigator.

The synonym file uses Tab-separated or comma-separated files only. If you have written your mapping requirement in any external application, for example, Microsoft Word or Microsoft Excel, you must export the relevant data in a format that the synonym file supports.

Item names in the file

A synonym file contains the names of items to be mapped, without the path to the item or the namespace of the item.

For example, if you want to map `partNum` to `partNumber` in the following XML, you must put `partNum` in the synonym file, not `item/partNum`, `items/item/partNum`, or `purchaseOrder/items/item/partNum`.

```
<po:purchaseOrder xmlns:po="http://www.ibm.com">
 <items>
    <item>
         <partnum>100-abc</partnum>
         <productName>Acme Integrator</productName>
         <quantity>22</quantity>
         <USPrice>100.99</USPrice>
         <po:comment>Acme Integrator</po:comment>
         <shipDate>2008-12-01</shipDate>
    </item>
 </items>
</po:purchaseOrder>
```

Synonyms in the file can:

- Be case sensitive or not case sensitive
- Contain the entire mapping item name
- Have non-alphanumeric characters removed

Rows in the synonym file

In the synonym file, each row represents one group of names to be mapped between each other and each row must contain at least two names. Names within a row are separated by commas in `.csv` files, and by Tab characters in `.txt` files.

A synonym file can contain an optional special row at the top. This top row contains key words **Input**, **Output**, or **Input_Output**, separated by the same delimiter used in the remainder of the file. The top row is used to indicate whether the synonyms are to be used to match names in mapping the input or the output:

- If the first word in the top row is **Output**, the first name only, in each subsequent row is searched in the mapping output for name matching.
- If the second word in the top row is **Input**, the second name only, in each subsequent row is searched in the mapping input for name matching.
- If the third word in the top row is **Input_Output**, the third name only, in each subsequent row is searched in both the mapping input and mapping output for name matching.

The top row must not contain fewer key words than the maximum number of names in any row in the file.

If the top row contains any word other than **Input**, **Output**, or **Input_Output**, the top row is ignored and it is assumed that the top row is missing. If you omit the optional top row, every name in the synonym file is considered to be **Input_Output**; that is, any name found either in the mapping input or in the mapping output is matched.

If a synonym file contains two rows:

```
car               automobile
automobile        vehicle
```

car and *vehicle* are not considered to be synonyms.

In order to make all three words synonyms, your synonym file can have either of the following structures:

- One row with all three words -

```
car          automobile          vehicle
```

- Three rows -

```
car          autombile
automobile   vehicle
car          vehicle
```

Special characters

You can write synonym files manually, or export them from another application; for example, Microsoft Excel.

Item names in synonym files reflect the application domain and do not have to match exactly the names in the XML schema or the relational database column.

For example, a synonym file might contain the row:

```
         summer       l'été
```

As `l'été` does not conform to the XML NCName format, you could name the element `l_été`. If all the alphanumeric characters in the synonym file match those in the schema, you can use the file with the option **Alphanumeric characters (Letters and digits only)**.

Many mapping requirements are written in Microsoft Excel, and cells in a Microsoft Excel file might contain specific characters like double quotation marks, space, new line, comma, and so on. When such a Microsoft Excel file is saved as a Tab-separated or comma-separated file, they contain additional double quotation marks.

Two groups of synonyms in a synonym file are delimited either by a Line Feed (LF) character, or Line Feed followed by a Carriage Return (LFCR). A Carriage Return (CR) character by itself does not end a group of synonyms.

Leading and trailing space characters adjacent to the delimiter (comma or Tab character) are ignored. Blank rows, or rows that contain only space characters, are permitted and ignored in a synonym file.

Different editors might inject different space characters into a synonym file; spaces are not used to delimit synonyms, and spaces are ignored unless they are inside double quotation marks.

If a synonym contains a comma, a double quotation mark, a carriage return, or a leading or trailing space that is significant, the synonym must be enclosed in double quotation marks. A double quotation mark within a synonym is escaped with another double quotation mark. For example:

```
        "comma,separated"
        "double""quote"
          "with<CR>
           newline"
          "  spaces   "
```

When the synonym file is read by the Graphical Data Mapping editor, the double quotation marks at the beginning and end of the synonym are removed and the following data is stored in the synonym table:

```
        comma,separated
        double"quote
          with<CR>newline
            spaces
```

The Graphical Data Mapping editor reads a synonym file containing these special characters correctly, and you should select the **Alphanumeric characters (Letters and digits only)** option when using the synonym file.

Algorithm used to match synonyms
The way in which synonyms are matched by the Auto map (automap) function, to create mappings between specific inputs and outputs, follows a set of rules.

1. Fixed value outputs and mapped outputs are excluded in name matching. Any output field that has a fixed value is excluded in name matching. Any output that is already mapped, or under a container that is already mapped, is excluded from name matching.
2. The synonym matching is done by using the name of an element, a database column, or an attribute regardless of its type. If an input and an output have the same name, it is a match, regardless of the kind of, and XSD type of, the input and output. An element, a database column, or an attribute can all form a match if their names are the same.
3. The synonym matching of alphanumeric characters is not case-sensitive, and is identical to that used in "Mapping input to output elements automatically" on page 1421.
4. Namespace or namespace prefixes do not participate in synonym matching. XML namespaces are excluded from name matching. Therefore, `abc:something` and `xyz:something` are considered the same, as are `{http://www.abc.com}:something` and `{http://www.xyz.com}:something`.
5. When multiple inputs have the `same` name as one output, one mapping is created. However, if you have multiple inputs with the same name as one output and you choose to map by the same name (or similar name) **and** to match descendants, an attempt is made first to match by path and name. If a match is found, one transform is made, and no further matches are considered.
6. If an input and an output have the `same` name, they are not considered a match under the option for synonyms. If you require a mapping for same-name inputs and outputs, you must also select the **Create transforms when the names of inputs and outputs are the same** option.
7. In addition to mapping synonyms, you might want to create mappings for some, but not all, same-name inputs and outputs. In this case, you have two options:
   - Clear **Create transforms when the names of inputs and outputs are the same**, and include the same-name inputs and outputs in the synonym file
   - Select **Create transforms when the names of inputs and outputs are the same**, and clear the unwanted mappings on the second page of the wizard.
8. When you select the **Map all simple descendants of the selected elements** option together with both same name and synonym mapping options, the following steps are taken to match names:
   - Compare the relative path and item name of the selected input and output
   - Compare the item name without relative path
   - Compare the item name without relative path to synonym

   Inputs and outputs matched in a previous step do not participate in later steps.
9. When you select the **Map the immediate children of the selected elements** option together with both same name and synonym mapping options, the following steps are taken to match names:
   a. Compare the item name without relative path
   b. Compare the item name without relative path to synonym

   Inputs and outputs matched in a previous step do not participate in later steps.

Creating and using a synonym file
You can use a synonym file to configure automatic mapping of input to output elements in the Auto map wizard (automap). You can use either a Tab delimited `.txt` file or a comma delimited `.csv` synonym file.

You can create a synonym file manually or generate a synonym file from the information that is contained in a Microsoft Excel spreadsheet.

Procedure

Complete the following steps to configure Auto map to use a synonym file that is created from the information that is contained in a Microsoft Excel spreadsheet:

1. Optional: Create a synonym file where the original mapping requirement is written in Microsoft Excel. The following set of instructions describe how to create a synonym file where the original mapping requirement is written in Microsoft Excel. If your original requirement is written in a table in Word, you must copy and paste the table into Microsoft Excel before you begin.
   a) Select the section of the Microsoft Excel spreadsheet that you require.
   For example, if you have a Product that you want to map to a Part number, select that section of the spreadsheet.
   b) Remove all columns from the spreadsheet, except the ones that contain the input field name and the output field name.
   You might have to edit some of the cells. For example, if your mapping instruction includes the phrase `based on`, remove this phrase.
   c) If the input or output fields contain paths, remove the paths to leave only the short names of the item.

   However, it is helpful to sort the column before you remove the paths. Sorted path names can indicate which is the best input or output to select when you start the action. If all the interested inputs or outputs start with the same path prefix, you might consider selecting the lowest input (or output) node in the tree, which has that common path prefix.
   d) Remove all rows that do not have an input field name and an output field name.
   For example, if you have an obsolete product that no longer has a part number and you have `n/a` in the input, remove that row.
   e) Select the **Save As** function in Microsoft Excel to save the spreadsheet into a format that is supported by IBM App Connect Enterprise Toolkit.

   You can use either a Tab delimited `.txt` file or a comma delimited `.csv` file. A comma delimited file can be opened with Microsoft Excel. The file can also be opened in a text editor.
2. Create the mappings by using the synonym file. Select the options in the Auto map wizard that match your requirements.
   For example, select the default options of **Map all simple descendants of the selected elements** and **Alphanumeric characters (Letters and digits only)**.

   When you choose these options, select **Create transforms when the input and output names are matched to synonyms defined in a file**.

   If you want to map same-name inputs to outputs, and the synonym file does not contain rows with those names (for example a row with **car,car**), select the **Create transforms when the names of inputs and outputs are the same** option, in addition to the **Create transforms when the input and output names are matched to synonyms defined in a file** option.

   You can select both **Create transforms when the names of inputs and outputs are the same** and **Create transforms when the names of inputs and outputs are more similar than**, in addition to **Create transforms when the input and output names are matched to synonyms defined in a file**, if your synonym file does not contain a row **color,color** and you want to map between them.
3. Click **Finish**.

Selecting matches
Use the Auto map wizard (automap) to select the mappings that you want to create.

About this task

When you have specified how you require the names to be matched on the initial panel of the Auto map wizard, and have selected **Next**, you see a panel that displays all the matches found.

You can now select the options that you require:

Procedure

1. Select a row in the **Transform Outputs** column that you want to change.
   Selecting a folder tree node results in the entire tree branch being selected or not selected.
2. Click **Edit** to start the **Select Transform Input** dialog.
3. To select a transform output, select the appropriate tree node check box.
   Conversely, to remove a mapping output, clear the appropriate tree node check box.
4. Ensure that you have selected only the number of matches that you require.
   The third column displays the number of transform inputs selected for each transform output. The cell has a value greater than one when the input of a transform contains several elements of certain names under various containers, and the input names match to the same output name.
5. Click **Finish** to complete the mapping process, or click **Back** to change the matches that you have set up.
   When you click **Finish**, you obtain a warning message if either, or both, of the following conditions apply:
   a. More than a few inputs to map to the same output.
   b. Many outputs for which you want to create mappings.

Specifying a transform (mapping operation)
Specify a transform, a cast function, or an XPath function between two or more elements by selecting from the list of available mapping operations that are shown on the connection.

Before you begin

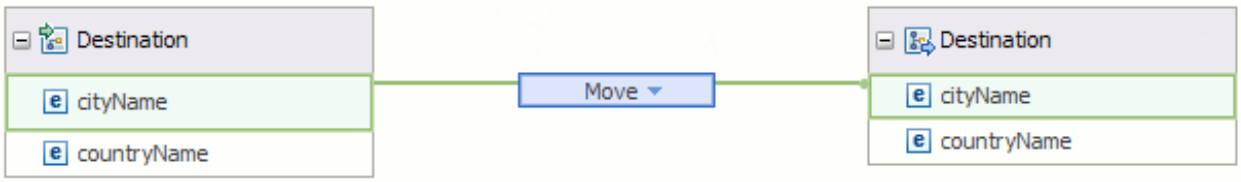
Learn about the different mapping operations. For more information, see "Transforms (Mapping operations)" on page 1280.

About this task

In the Graphical Data Mapping editor, you can use transforms, cast functions, and XPath 2.0 functions to run different actions on input data and move the result to the output element. You choose the appropriate mapping operation that is based on the result that you want to achieve.

When you create a connection between two or more elements, a transform is assigned, based on the number and type of input elements. You can then change the transform by choosing from a list of available transforms. If a particular transform type is not shown in the list, that transform is not valid for your input and output elements.

For example:



Note: Transforms that require multiple inputs such as **ForEach**, **Join**, or **Append** are not available in the list of available transforms until you wire two inputs to the transform.

Note: When you drag and drop a connection from an input element to an output element, a transform is added automatically. To add the transform, you must drop the connection above the lowest point of the

input and output message. If you use this method in a nested map, you must drop the connection above the bottom of the shaded area.

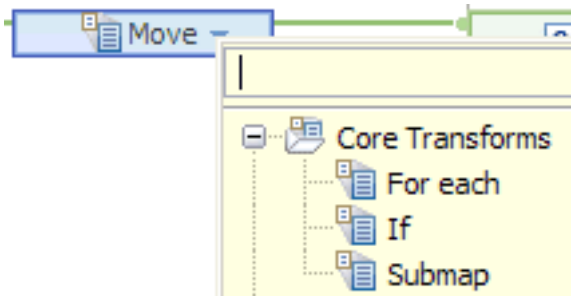
When you have a list of valid transform types, choose the appropriate transform:

- If you have a single array as input, with the same array type as output, and you want to move all elements to the output, use **Move**.
- If you have a single array as input, and you want to iterate over each element in the array (for example, you might want to remove some elements) use the **ForEach** transform and set the cardinality options.
- If you have multiple input elements, you can use the **Append** or the **Join** transforms. If you use the **Append** transform, the number of output elements is the total of the input elements. If you use **Join** transform, the number of output elements depends on the user expression added to specify the matching criteria for joining or filtering input items.

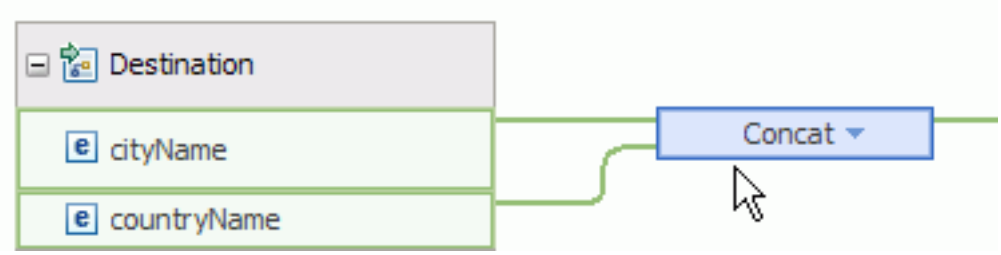
The following steps show how to change the transform that is selected, and also how to add more elements:

Procedure

1. Change the transform by clicking the arrow in the transform box, and selecting from the list of available transforms.
   For example:



2. If you create more primary connections between an input element and the transform, the transform type changes to one that allows multiple primary inputs, such as **Local Map**, **Concat**, or **Join**.
   For example:



What to do next

Configure the properties of the transform. For more information, see “Configuring the properties of a transform” on page 1431.

Configuring the properties of a transform
Configure the properties of a transform to set the value of an output element; define a conditional expression that determines whether the transform is applied; define the indexes to use when the input

element is a repeating structure; or reorder the way in which inputs to a transform are handled by the Graphical Data Mapping editor.

Procedure

Complete the following steps to configure the general properties of a transform:

- Open the **Properties** view of a transform by using any of the following methods:
  - Select a transform. Then, select the **Properties** tab.
  - Right-click the transform, and then select **Show in** > **Properties view**.
  - When the map is in full vie, select `Alt+Enter`.

    You can configure these keyboard shortcuts by selecting **Window** > **Preferences** > **General** > **Keys**.

  Note: You can detach the **Properties** view from the map. This is very useful if you have a secondary workstation screen. In one workstation screen you can see the map, and in the other you can see the properties map.
- In the **Properties** tab, add, modify, or remove resources:
  - Select **Cardinality** to select the indexes of the input and output elements that you want this transform to operate over. For more information, see "Selecting the indexes of input array elements" on page 1363.
  - Select **Variables** to list the names of the input elements that are connected to the transform. Select **Edit** to change the name of a variable.

    Note: When you change the name of a variable, the new name is the one you see when you use content-assist to create your expressions.
  - Select **Condition** to define the XPath expression that must be evaluated against the input element before the transform is applied. If the condition evaluates to **true**, the transform is applied. For more information, see "Defining an XPath conditional expression for a transform" on page 1433 and "Defining a Java conditional expression for a transform" on page 1439.
  - Select **Filter** to define an expression that must be evaluated against each element on a repeating input element before the transform is applied. If the condition evaluates to **true**, the transform is applied for each input.
  - Select **Sort** to sort the inputs to the transform by ascending order, descending order, case order, or data order. Then, complete the following steps:

    1. Select **Sort the inputs to this transform**.
    2. Add elements to the **Sort by** column.
    3. Select a sort method.

    Note: The element or elements that are used to provide the sort must provide a singular, scalar value. If you nest sorts, each sort uses the original input data value for the sort element. Specifically, the outer sort cannot use the result of a nested sort.
  - Select **Order** to display the order of input connections to a transform. You can reorder them.
  - Select **Documentation** to provide a description of the transform, or other relevant usage notes.

What to do next

Deploy and test the message map. For more information, see "Troubleshooting a message map" on page 1587.

Defining an XPath conditional expression for a transform
You can define an XPath expression to set the conditional expression that determines whether a transform is applied in a message map. When the XPath expression evaluates to true, the transform is applied.

About this task

When you define an XPath expression, you use the variable names of input elements. The value of an input element has an *effective Boolean value*.

The effective Boolean value is false for the following input elements:

- The input element is Boolean and its value is `false`.
- The input element is a sequence, and the sequence is empty.
- The input element is string and its value is the empty string `""`.
- The input element is a float or a decimal and the value is `NaN (not a number)`.
- The input element is of a numeric type and its value is `0`.

In any of these cases, the XPath expression can be formed from just the variable name that represents the input element.

You can also get a Boolean result from defining expressions that use variable names that represent inputs and constant values and any of the following operators :

- Logical operators such as `and`, `or`, `not`.
- Comparison operators such as =, `!=`, <, >, <=, >=.

Note: Always use content assist to select the variable name of the input elements that you use to define the XPath expressions. If you do not use content assist, you may be using an incorrect element name and your map will fail at run time.

Note: XPath 1.0 functions are valid XPath 2.0 expressions. You can use the XPath Expression Builder to generate simple XPath 1.0 expressions.

Note: If you define a conditional expression for a transform, and then realize that the condition is identical for other transforms in the map, change the transform to an **If** transform. The conditional expression remains unchanged.

Procedure

Complete any of the following steps to set a conditional expression in a transform:

- For non-repeating elements, select the **Condition** property in the **Properties** tab, and enter an XPath expression.
- For repeating elements, select the **Filter Inputs** property in the **Properties** tab, and enter an XPath expression that will be applied to each instance of the repeating element.

  For more information, see “Defining an XPath conditional expression for a structural transform (ForEach)” on page 1437.

Results

The input element is evaluated against the condition. If the condition evaluates to true, the transform is applied to the input element.

Example

The following examples show how to define simple XPath conditional expression for a transform when you have non-repeating elements:

Example: XPath expression to check a Boolean input element

This example shows how to define the XPath expression for a Boolean element so that it evaluates to true. The expression depends on the value of the Boolean element.

The XML schema for the element is the following:

```
<element name="IsEmployee" type="boolean"></element>
```

When the value of a Boolean element is set to **false**, the XPath expression that you define is the following:

```
fn:not($IsEmployee)
```

When the value of the Boolean element is set to **true**, the XPath expression that you define is the following:

```
$IsEmployee=fn:true()
```

The $IsEmployee expression on its own tests if the element exists and always evaluates to true if the element exists regardless of what value it is set to.

Example: XPath expression to check if the value of a numerical input element is greater than a constant value

This example shows how to define an XPath expression that evaluates to true when the value of a numerical element is greater than 200.

The XML schema for the element is the following:

```
<element name="BusinessUnit" type="int" ></element>
```

The XPath expression that you define is the following:

```
$BusinessUnit > 200
```

Example: XPath expression to check if a numerical input element has a specific value

This example shows how to define an XPath expression that evaluates to true when the numerical input element has a value of `0`.

The XML schema for the element is the following:

```
<element name="QtyBooks" type="int" ></element>
```

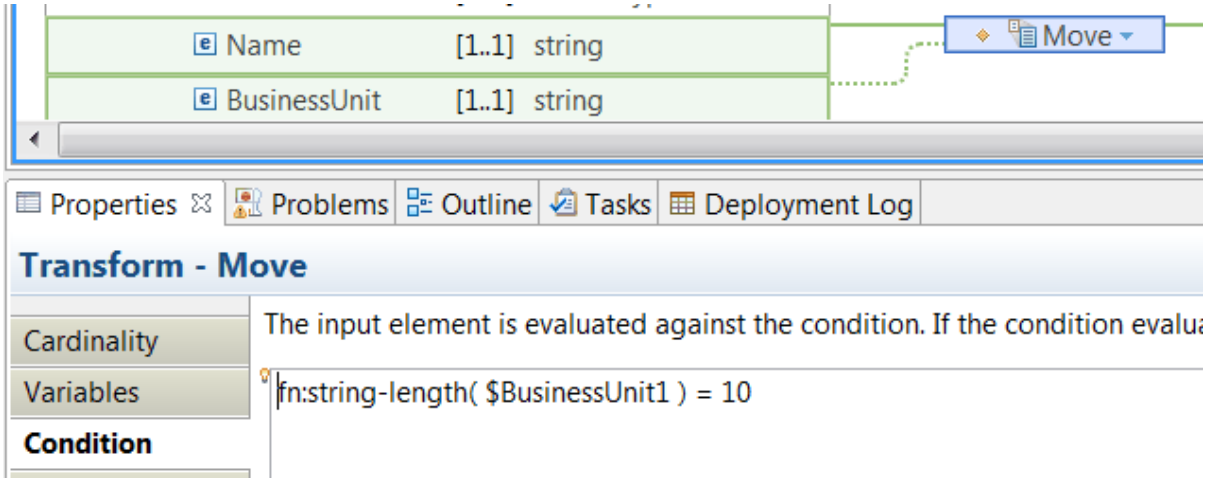
The XPath expression that you define is the following:

```
$QtyBooks = 0
```

Example: XPath expression to check the string length of an input element

This example shows how to define an XPath expression that evaluates to true when the length of a string is 10.

- The **Move** transform has a primary connection wired from the element **Name**.
- The **Move** transform has a secondary connection wired from the element **BusinessUnit**. This connection is supplementary because this element is not used to calculate the value of the output element.
- The **Move** transform should only execute if the length of the element **BusinessUnit** is `10`.



The XML schema for the elements are the following:

```
<element name="Name" type="string" ></element>
```

```
<element name="BusinessUnit" type="string" ></element>
```

The XPath expression that you define is the following:

```
fn:string-length($BusinessUnit) = 10
```

Example: XPath expression to check if a string input element is set to the empty string

This example shows how to define an XPath expression that evaluates to true when a string element is empty.

To define an XPath expression that checks if a string element is empty, you must use the operator `!=`.

The XML schema for the element is the following:

```
<element name="Name" type="string" ></element>
```

The XPath expression that you define is the following:

```
$Name != ''
```

Example: XPath expression to check if a repeating element is empty

This example shows how to define an XPath expression that evaluates to true when a repeating element, which is referred to as a sequence, is empty.

The effective Boolean value of an empty sequence is false. You could use `fn:not($Address)`, but it is easier to understand if you use `fn:empty()`.

The XPath function **fn:empty()** evaluates to `true` if a sequence is empty.

The XML schema for the element is the following:

```
<xsd:element form="qualified" name="Address" type="mqsistr:Address"
maxOccurs="unbounded" minOccurs="0" />
```

The XPath expression that you define is the following:

```
fn:empty($Address)
```

Example: XPath expression to check if an optional input element is present

This example shows how to define an XPath expression that evaluates to true when an optional element is present.

Use the `fn:exists` XPath function if the input element is a boolean. Otherwise, you can use the effective Boolean value.

The XML schema for the element is the following:

```
<element name="BookName" type="string" maxOccurs="unbounded" minOccurs="0" ></
element>
```

The XPath expression that you define is the following:

```
$BookName
```

Example: XPath expression to check if a complex input element has no content, that is, it is empty

To determine whether a complex element is empty, you must check for the presence of child elements or attributes.

Testing the effective Boolean value of elements or attributes that are present in a complex type will yield true. You can use the `fn:not` XPath function to invert a boolean.

The `fn:not` function accepts a sequence of items. The value that this function returns is true if any of the arguments is either a single Boolean value false, a zero-length string, the number 0 or NaN, or the empty sequence. Otherwise, it returns false.

This example shows how to define an XPath expression that evaluates to true when the complex input element has no children.

The XML schema for the elements are the following:

```
<complexType name="Address">
    <sequence>
 <element name="Type" type="string"/>
 <element name="Number" type="integer"/>
 <element name="Street" type="string"/>
 <element name="Postcode" type="string"/>
 <element name="City" type="string" />
 <element name="Country" type="string"/>
 <element name="AdditionalInfo" type="string"/>
    </sequence>
</complexType>
```

The XPath expression that you define to check for child elements being present in a complex element is the following:

```
fn:not($Address/* )
```

The XPath expression that you define to check for attributes being present in a complex element is the following:

```
fn:not($Address/@* )
```

The XPath expression that you define to check for elements and attributes being present in a complex element is the following:

```
fn:not($Address/*) and fn:not($Address/@* )
```

What to do next

Deploy and test the message map. For more information, see “Troubleshooting a message map” on page 1587.

Defining an XPath conditional expression for a structural transform (ForEach)
Configure the **Filter Inputs** section in the **Properties** view of the **ForEach** transform to define the conditional expression that determines whether the transform is applied in a message map.

About this task

The **ForEach** transform can only have one **primary** input connection. Additional connections to the **ForEach** transform must be of type **Supplement**.

Procedure

Complete the following steps to define a conditional expression on a **ForEach** transform:

- Select **Allow empty input** if you want the **ForEach** transform to execute at least one.

  If you select this option, the transformations that you define in the **ForEach** transform nested map will execute once regardless of the conditional expression.
- Define the XPath expression that determines whether the **ForEach** transform is applied in the map.

  Note: Always use content assist to select the name of the input elements that you use to define the XPath expressions.

  The conditional expression applies to all the indexes that you configure in the **Cardinality** tab of the **ForEach** transform properties view.

Results

The input element is evaluated against the condition. If the condition evaluates to true, the transform is applied to the input element.

Example

This example shows how to define an XPath expression that checks the value of a string element:

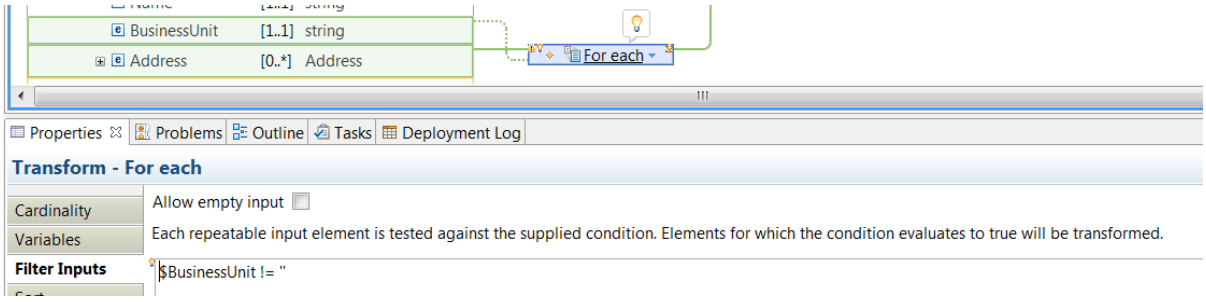
- The **ForEach** transform has a primary connection wired from the repeating element **Address**.
- The **ForEach** transform has a secondary connection wired from the mandatory element **BusinessUnit**. This element can be set to nil. This element is not used to calculate the value of the output element.
- The **ForEach** transform should only execute if the **BusinessUnit** element is not empty.

The XML schema for the mandatory element is the following:

```
<element name="BusinessUnit" type="string" nillable="true"></element>
```

The XML schema for the repeating element is the following:

```
<xsd:element form="qualified" name="Address" type="mqsistr:Address"
maxOccurs="unbounded" minOccurs="0" />
```



The XPath expression that you define is the following:

```
$BusinessUnit != ''
```

What to do next

Deploy and test the message map. For more information, see "Troubleshooting a message map" on page 1587.

Choosing an XPath conditional expression that tests for a nil value in a transform
Use the XPath functions **fn:empty**, **fn:nilled**, or **fn:exists** to test if an output element is set to nil. You can use these functions to define conditional expressions in a transform.

About this task

For example, you can use **fn:empty**, **fn:nilled**, or **fn:exists** as part of conditional expression that determines if a transform is applied. You can also use these transforms as part of the conditional expressions that you set in an **IF** transform.

Procedure

- Choose any of the following XPath functions to determine the state of a source element:

XPath function	Usage	Source element	Output
fn:empty	Tests whether a set of elements is empty	A single XML or non-XML element that is present and nilled	false
fn:empty	Tests whether a set of elements is empty	A NULL value in a logical tree	false
fn:empty	Tests whether a set of elements is empty	A missing element, not present in the logical tree	true
fn:nilled	Tests whether an element is nilled	A nillable XML element	true only if the xsi:nil attribute is present and set to 'true' in the logical tree
fn:nilled	Tests whether an element is nilled	A nillable non-XML element	true only if the value in the logical tree is NULL
fn:exists	Tests whether an element exists	A single XML or non-XML element	true if the element is present in the logical tree, regardless of the nillable and nilled state
fn:exists	Tests whether an element exists	A NULL value in a logical tree	true
fn:exists	Tests whether an element exists	A missing element, not present in the logical tree	false

Example

This example shows an XPath expression that checks if an input element is nilled.

The XPath expression evaluates to true when an input element is not set to nil.

Use the `fn:nilled` XPath function to test whether the value of an input element has the `xsi:nil` attribute set.

Note: An XML element that has the `xsi:nil` attribute set is considered to be present.

The XML schema for the element is the following:

```
<element name="BookName" type="string" nillable="true" ></element>
```

The XPath expression that you define is the following:

```
fn:nilled( $BookName)
```

Defining a Java conditional expression for a transform
You can use a Java method to set the conditional expression that determines whether a transform is applied in a message map.

About this task

To define the conditional expression, you can define an XPath expression or a call to a static method on an imported Java class. You can also create a compound (nested) expression that includes XPath, Java and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

Note: When you define a Java conditional expression, the following rules apply:

- You can use Java methods that only use standard Java types.
- You cannot use Java methods that include **MbElement** data type arguments.
- You can use Java methods that use the **DOM API**. For more information on the supported types, see "Custom Java" on page 1312.

Note: If you define a conditional expression for a transform, and then realize that the condition is identical for other transforms in the map, change the transform to an **If** transform. The conditional expression remains unchanged, and the transform is moved within the nested level of the **If** transform. Then, place all of the transforms that depend on the same condition in the **If** nested mapping.

Procedure

Complete the following steps to set a conditional expression in a transform:

1. Create a message map.

   When you use Java code in a map, you can define the map in a static library project, a shared library project, an application project, or an integration project. You can define the Java code in one or more Java projects. Your map project must reference these Java projects. For more information, see Adding a project to an application, integration service, or library

   Note: When the map uses Java code available in a shared library, you must create the map in the shared library project where the Java code is available.

   For more information, see "Creating message maps" on page 1378.
2. Create a Java project.

   For more information, see Creating a Java project.
3. Create Java object classes.

   The Java class that you provide to the map must have static methods that return Boolean values.

   For example, the following Java method might be used to define a conditional expression on a transform:

```
public static boolean lessThanHund(BigDecimal val) {
  if ( val.intValueExact() < 100 )
   return true;
```

```
  return false;
}
```

4. Ensure that the Java project the class is in is referenced by the project that contains the map.

   Before you create a reference to a Java project, ensure that the Java class is available in a Java project in your workspace.

   a. In the **Application development** page, right-click the project where the map is defined.

   b. Select **Managed included projects**.

   c. Select the Java project, and then click **OK**.

5. Make the Java class visible anywhere in the map. Unless you have already used a method from the Java class in a Custom Java transform elsewhere in the map, you must manually create an import for the class. To manually import the Java class, you must configure the **Java imports** tab in the **Properties** page of the map:

   a) Select **Add**.

   b) Click **Browse**.

   c) In the **Select entries** field, enter the name of the Java package that contains the Java class.

      Note: You must start typing the name of your Java package before you can see any classes to choose from in the **Select entries** field.

   d) Select a Java class.

   e) Click **OK**.

   f) Enter a unique value in the **Prefix** field.

   Note: If the Java project that contains the Java class does not build in Eclipse, then the Java class is not visible anywhere in the map. You must resolve all the Java errors before you can configure the **Java imports** tab in the **Properties** page of the map. You can see the errors in the **Problems** tab.

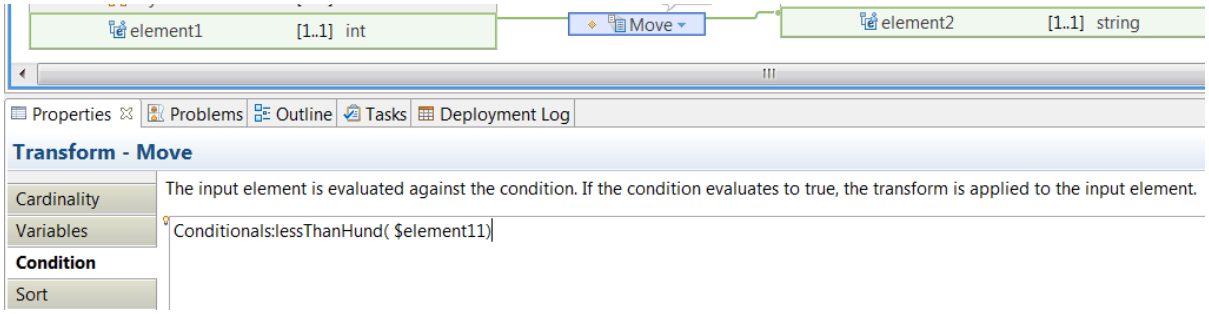
   When you add a Java call in the **General** tab of the **Properties** page of a **Custom Java** transform in your map, an import is automatically added in the map to refer to the package qualified Java class, and a prefix based on the class name is added. All the **public static methods** in this Java class are then available in content assist when building expressions in other transforms. If you want to use this class, you can skip this step.

6. For non-repeating elements, select the **Condition** property in the **Properties** tab, and enter a Java method.

   Note: Always use content assist to select the variable name of the input elements that you use to define the expressions. If you do not use content assist, you might be using an incorrect element name and your map fails at run time.

   You can identify the Java method in content assist by using the prefix. For example, in the following figure the prefix is **Conditionals** and the method is **lessThanHund()**:

   ```
   Conditionals:lessThanHund( $element11)
   ```

   Note: If the input element being used to provide a value for a Java method is not of the correct type, you can use a type cast function, for example **xs:int($var)**, to set the required type. For more information, see "Cast type (xs:type)" on page 1299.
7. For repeating elements, select the **Filter Inputs** property in the **Properties** tab, and enter a Java method that is applied to each instance of the repeating element.

Results

If the Java method returns true, the transform is applied to the input element.

Example

You can call the following Java method when you define a conditional expression in a map:

```
public static boolean evalNodeCondition(Node inputNode, String hasChildNamePrefixed)
    { boolean result = false;
      if ( inputNode != null ) {
            Node fcn = inputNode.getFirstChild();
            if ( fcn != null ) { result = fcn.getLocalName().startsWith(hasChildNamePrefixed);
            }
     }
return result;
```

What to do next

Deploy and test the message map. For more information, see "Troubleshooting a message map" on page 1587.

Grouping transforms into nested maps
Use the **Group transforms into nested maps** property to nest the transforms in a new **Local map** transform. The local map provides a way of displaying parts of a larger message map, enabling you to view the message map elements in a hierarchical way.

About this task

When the **Group transforms into nested maps** property is selected, the Auto map wizard (automap) attempts to place new transforms inside new message map transforms.

The Auto map wizard recursively analyzes the input and output elements of the new transforms, searching for a common input ancestor and a common output ancestor. If a common input ancestor is found for the input elements, and a common output ancestor is found for the output elements, a new **Local map** transform is created between them. Inside the new local map, when no more common ancestors can be found, the new transforms are created at this level. If there are still some common ancestors at a given level, the process is repeated.

For example, a map might contain the following input:

```
SomeInput
--Resources
---field1
---field2
---field3
---field4
```

and the following output:

```
SomeOutput
---Items
----field1
----field2
----field3
----field4
```

If the **Group transforms into nested maps** property is not selected, the Auto map wizard creates transforms between the input and output elements `field1`, `field2`, `field3`, and `field4` at the current level of the map.

However, if the **Group transforms into nested maps** property is selected, the Auto map wizard creates a **Local map** transform between `SomeInput/Resources` and `SomeOutput/Items` at the current level of the map, and then creates transforms between the input and output elements `field1`, `field2`, `field3`, and `field4` inside the local map.

Moving transforms into a local map or a submap
In the Graphical Data Mapping editor, you can group transforms into a local map or submap to enhance readability and maintenance of the message map. You can re-factor a local map to a submap to enable reuse of mappings of global complex types in multiple maps.

About this task

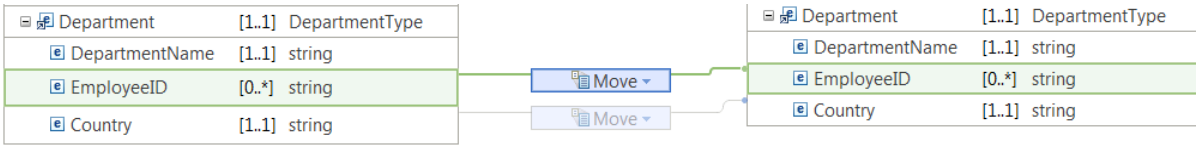
You can use a **Quick Fix** to move one or more transforms inside a local map or submap.

Your map has one or more transforms that are defined between child elements of an input and an output complex element. Then, you decide to change the design of your map, and move these transforms into a local map or into a submap.

Procedure

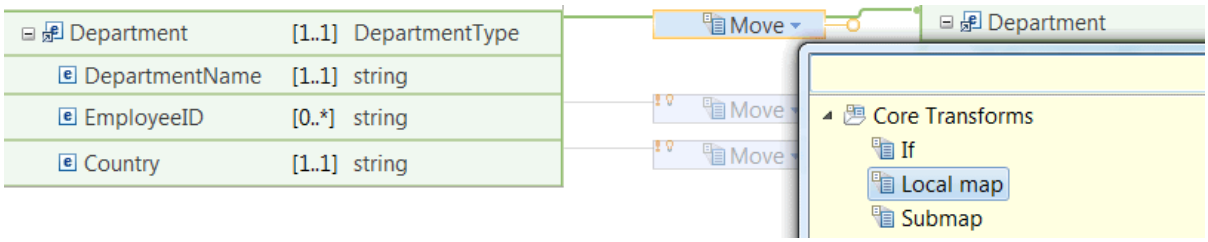
You must complete the following steps to move one or more transforms into a local map or submap:

1. Create a map with a complex input element and a complex output element.
2. Define transforms between the input and output child elements.



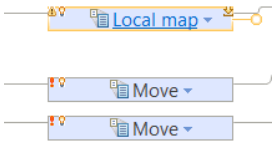
3. Change map design to group part of the transforms in a local map or submap.
4. Wire a **Move** transform from the source folder to the target folder.

5. Change the **Move** transform to a **Local map** transform or **Submap** transform.

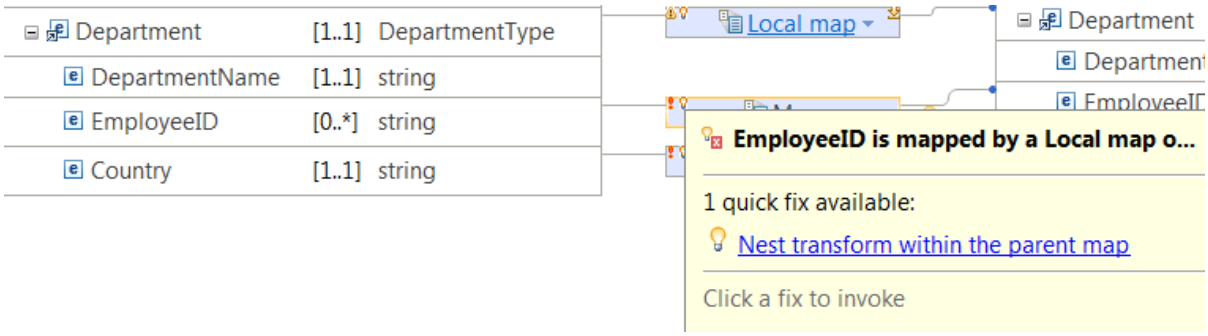


The Graphical Data Mapping editor displays an error marker on each child field mapping. This error indicates that the element is also mapped at the parent level.

The Graphical Data Mapping editor offers a **Quik Fix** on each transform to move the transform inside the local map or submap.



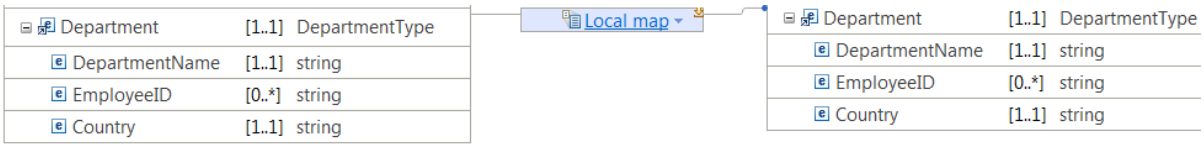
6. Click the **Quick Fix** associated to each of the transforms that you want to move inside the local map or submap. Then, click **Nest transform within the parent map**.



The local transform moves inside the local map. The Graphical Data Mapping editor automatically re-factors the map.

Results

The transforms, that initially were defined at the main level, are now located inside the local map or submap.



What to do next

Deploy the message map and verify that the output message is valid. For more information, see “Troubleshooting a message map” on page 1587.

Using content assist (Mapping syntax)
Use content assist to find the variable names for input and output objects, database elements, and paths.

Press **Ctrl+Spacebar** to display content assist, and use it to help you build your XPath statements and mapping expressions. Content assist provides the variable names that you need to use to reference

elements in your XPath statements and expressions. The variable name assigned for the same element can vary between uses in different transforms, so always use content assist to obtain the correct variable name. Avoid copying and pasting expressions that include variable names.

You can use content assist for the following tasks:

- Making comparisons
- Performing arithmetic
- Creating complex conditions

The comparison operators are:

= (equals)
!= (not equals)
> (greater than)
>= (greater than or equal)
< (less than)
<= (less than or equals)

The arithmetic operators are:

+ (plus)
- (minus)
* (multiply)
div (divide)

For information about XPath syntax, see W3C XML Path Language (XPath) 2.0.

Conditional operators (`or` and `and`) are supported; these are case-sensitive.

The following objects can be mapped:

- Message elements (defined in the schema for the input and output)
- Message assembly, comprising the properties tree and optionally the LocalEnvironment and transport headers
- Data from database tables

Database objects with names that do not conform to the XML NCName format

Some database objects have names that do not conform to the XML NCName format (for example, the name contains characters like #, or $). If the database object name is used in SQL (for example, in the `where` clause of the **Select** transform), no action is required. If the database object is used in XPath (for example, in a Custom transform or a condition), use content assist, which adds the appropriate XPath-compliant expression.

Deleting objects and transforms
You can delete input objects, output objects, and transforms from a message map by using the Graphical Data Mapping editor.

Procedure

- To delete an input object or an output object from a message map, choose any of the following methods:
  - Select the appropriate object in the Graphical Data Mapping editor, and then click the **Delete selected elements** icon in the toolbar.
  - Select the object and then either press the delete key **Del**, or right-click and select **Delete** from the menu.

  You can delete multiple elements at the same time by marking them by using **Ctrl + click**, and then clicking the **Delete selected elements** icon.

- To delete a transform from a graphical data map, choose any of the following methods:
  - Select the transform and press the delete key (**Del**).
  - Select the transform, right-click on the transform, and then select **Delete** from the menu.

What to do next

Continue editing the map, and define transformations between the input message and the output message. For more information, see "Editing message maps" on page 1394.

Advanced editing in a message map

You can configure the input and output message assembly in a message map by using the Graphical Data Mapping editor.

Procedure

- Configure the general properties of the input and output message assembly by using the Graphical Data Mapping editor. For more information, see "Configuring the properties of the input and the output message assembly to a message map" on page 1445.
- Optional: Configure the message assembly to include any of the following components: the Properties tree, one or more message tree headers, the message tree body, and the local environment tree. For more information, see "Configuring the message map to include message assembly components" on page 1447.

  In IBM App Connect Enterprise, the message assembly is the internal representation of a message. When you transform a message, you might need access to elements in a message assembly component or you might need to modify some of these elements in your message map.
- Transform transport headers. For more information, see "Mapping transport headers" on page 1461.
- Transform elements in the Properties tree by using transforms such as the **Move** transform to copy a value, or the **Assign** transform to set a value of an element. You can also use elements in the Properties tree as input data to your transformations. For more information, see "Mapping the Properties tree" on page 1463.
- Update, delete, or create data in the local environment tree. You can use the environment tree as input data to your transformations. For more information, see "Mapping data in the local environment tree" on page 1464.
- Update, delete, or create data in the environment tree. You can use the environment tree as input data to your transformations. For more information, see "Mapping the environment tree" on page 1471.
- Configure a database to map or modify database content. For more information, see "Adding database definitions to the IBM App Connect Enterprise Toolkit" on page 1474.
- Use a Mapping node to access properties that are associated with the message flow that contains the node. For more information, "Accessing user-defined properties from a Mapping node" on page 1479.

What to do next

Configure mapping transforms in your message map. For more information, see "Choosing a transform to set the value of a simple type output element" on page 1358.

Configuring the properties of the input and the output message assembly to a message map

You can configure the general properties of the input and output message assembly in a message map by using the Graphical Data Mapping editor.

About this task

The input and output message assembly properties are located within the **General** tab. They provide information that is displayed for information only, and cannot be modified. In addition, you can configure

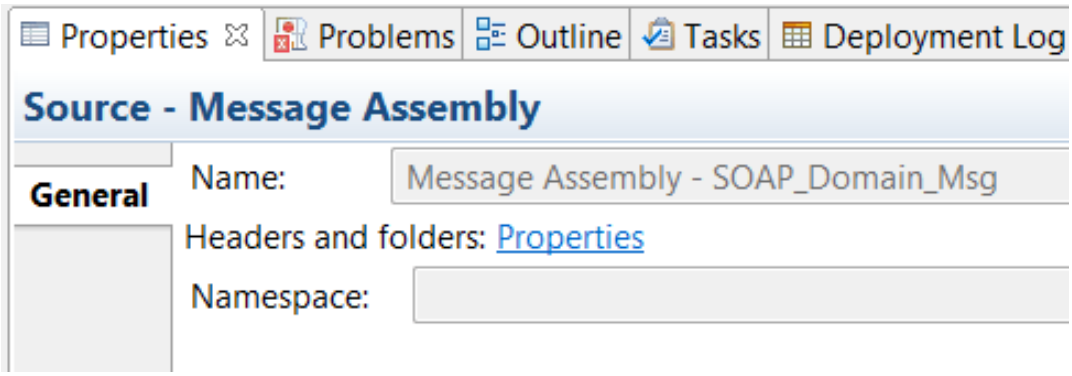
the message assembly components that you might require to define your data transformations, and the message domain that defines the serializer to convert the logical tree structure into a bit stream.

Procedure

Complete the following steps to configure the general properties of the input and output message assembly to a message map:

- Configure the source message assembly components:
  a) Select the input message assembly in the map, and then select the **Properties** tab.

     You can see the **Source - Message Assembly** properties:

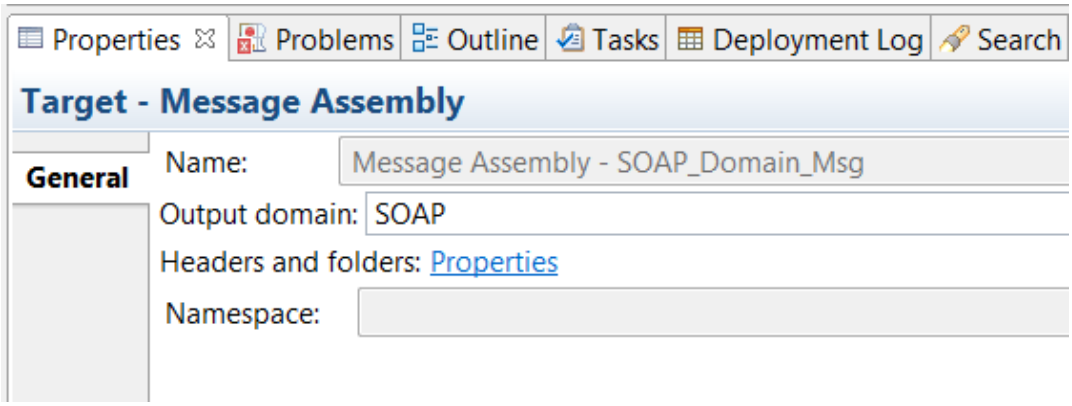


  b) Add or remove message assembly components to the message map source message assembly.

     For more information, see "Customizing a message map to include a message assembly component" on page 1458.

- Configure the target message assembly components:
  a) Select an output message assembly in the map, and then select the **Properties** tab.

     You can see the **Target - Message Assembly** properties:



  b) Specify the **Output domain**. This property defined the parser used to create the output message.
  c) Add or remove message assembly components to the message map target message assembly.

     For more information, see "Customizing a message map to include a message assembly component" on page 1458.

What to do next

Define the map transformations, For more information, see "Specifying a transform (mapping operation)" on page 1430.

Configuring the message map to include message assembly components
In IBM App Connect Enterprise, the message assembly is the internal representation of a message. When you transform a message, you might need access to elements in a message assembly component or you might need to modify some of these elements in your message map. You can configure a message map to include the following message assembly components: the Environment tree, the message tree Properties tree, message tree headers, the message tree body, and the local environment tree.

About this task

When a message arrives to an application or to an integration service, it is received by an input node that you have configured in a message flow. Before the message can be processed by the message flow, the message must be interpreted by one or more parsers that create a logical tree representation from the bit stream of the message data. The logical tree is also known as the message assembly. The tree format contains identical content to the bit stream from which it is created, but it is easier to manipulate in the message flow.

Procedure

To include message assembly components into your message map, complete the following steps:

1. Identify the message assembly components that you need to add to your message map. For more information, see "Choosing message assembly components to include in a message map" on page 1448.

   You may need to include the local environment tree to use information provided in a variable or you may need to add a header to access transport specific information.

   You can store information in the environment tree while the message passes through the message flow. You might need to use data passed in the environment tree to calculate the value of output elements in a message map. You might need to use data from the environment tree to determine whether a transform should be applied.
2. Identify whether you need to initialize, delete, or transform elements in components of the message tree or in the local environment tree. For more information, see "Choosing a mapping action" on page 1450.

   You can add different parts of the message tree to the map input, to the map output, or to both. You can also add the local environment tree. Depending on how you add a message assembly component, this component can be deleted, initialized, or transformed.
3. Configure the message map to include a message assembly component. For more information, see "Customizing a message map to include a message assembly component" on page 1458.

   To customize your message map to include more message assembly components, you must add message assembly components to the input message and to the output message, and then define transforms between them.
4. Configure the message map to include the environment tree.

   For more information about mapping the environment tree, see "Mapping the environment tree" on page 1471.

Results

You now have a message map that includes the message assembly components that you need to complete your message transformation.

What to do next

Define transforms between other message assembly components that you have included. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Choosing message assembly components to include in a message map
You can add the message tree components, that is, the Properties tree, the headers, and the message body into a message map. You can also add the local environment tree into a message map, and the Environment tree.

About this task

In IBM App Connect Enterprise, the logical tree structure is the internal representation of a message. The logical tree structure is created by the parser when the message is received by an input node. It is also known as the Message tree and makes up part of the message assembly. The message assembly consists of four trees:

- Message tree: This tree includes the Properties folder, the message body, and headers.
- Environment tree
- Local environment tree: This tree includes multiple destination folders, and a variables folder.
- Exception list tree

When you create a message map, the Properties folder and the message body are automatically included in your Graphical Data Mapping editor.

Note: You can remove the properties folder and the message body in the message map if you only want to modify the local environment tree. This will accelerate your message transformation since the message properties and the message body will be copied over without the need to bring them into the transformation engine.

You cannot add the exception list tree to the message map. The exception list is included automatically, and the entire contents of the input exception list is retained in the output.

Procedure

Choose one or more message assembly components to include in a message map:

- **Header folders**: You can add one or more headers to a message map, in addition to the Properties folder and the message body.

  When an input message is received by an input node, the input node invokes the correct parser for each header, and includes in the message tree the corresponding headers. You can then access these headers by using message maps.

  The message tree always includes the following components:

  – All the headers that are present in the message.
  – The message body.
  – The Properties folder. The Properties folder (sometimes referred as the Properties tree) is the first element of the message tree and holds information about the characteristics of the message. When the input node receives the input message, it creates and completes the Properties folder.

  If you need to access information available in an element of a header or if you need to modify it, then you must add the header to the message map. For more information, see “Mapping transport headers” on page 1461.

- **Local environment tree**: You can add the local environment tree to the message map. The local environment is divided into two parts:
  - Standard folders that are automatically defined for each of the destination folders available in IBM App Connect Enterprise.
  - A variables folder that is added automatically. You can use the **Cast** function to include a variable into your message map. You can add variables by using the **Add User Defined** function.

  The local environment tree stores variables that can be referred to and updated by message processing nodes that occur later in the message flow.

  You can also use the local environment tree to define where a message is sent. The destination can be internal or external to the message flow.

  IBM App Connect Enterprise also stores information in the local environment tree in some circumstances, and references it to access destination values that you might have set.

  If you need to access information available in an element of the variables folder or if you need to modify a variable, then you must add each individual variable to the message map. For more information, see "Configuring a generic type in the local environment tree by using the Cast function" on page 1466.
- You can add the environment tree to the input message assembly of the message map.

  If you do not include the environment tree in the input message assembly, the entire contents of the environment tree are retained in the output.

  If you add the environment tree, you can define both its content and structure by using the **Add User Defined** function.

  Note: The environment tree differs from the local environment tree in that a single instance of it is maintained throughout the message flow.

Results

The following table summarizes the message assembly tree folders that you can include into your message map:

Table 57. Message assembly trees that can be included in a message map

Message assembly trees	Folders in a message assembly tree	Can be configured in a message map as an input to the map and as an output to the map?	Status in a message map
Message tree	Properties folder	Yes	Required
Message tree	Header folders	Yes	Optional
Message tree	Message body	Yes **Note:** You must cast parts of the SOAP message body to be able to define transforms between its input and output elements.	Required
Local environment tree	Variables folder	Yes (You must cast a variable to define transforms between its input and output elements.)	Optional
Local environment tree	Destination folders	Yes	Optional

Table 57. Message assembly trees that can be included in a message map (continued)			
Message assembly trees	**Folders in a message assembly tree**	**Can be configured in a message map as an input to the map and as an output to the map?**	**Status in a message map**
Environment tree		Yes	Optional
Exception list tree		No	

What to do next

Identify the configuration of the different message assembly components. For more information, see "Choosing a mapping action" on page 1450.

Choosing a mapping action
You can add different parts of the message tree to the map input, to the map output, or to both. You can also add the local environment tree. Depending on how you add a message assembly component, this component can be deleted, initialized, or transformed.

Procedure

Identify the action that you want to achieve in your message map to find out how to add a message assembly component into the message map:

- To copy a message assembly component unchanged, do not include the component in the message map.

  The mapping engine copies the local environment tree and any other headers and folders from input to output, unchanged, when they are not included in the message map.

  The mapping engine handles the following components as independent units of transformation:

  – The properties tree in the message tree
  – The message body in the message tree
  – Each header structure in the message tree. For example, the MQMD is treated as one unit, the MQRFH2 header is considered an independent unit, and so on.
  – The local environment tree

  If any of these units is not included in the message map, the mapping engine copies their contents unchanged.

  Note: If the only transform that you define between an input map component and an output map component is the **Move** transform so that the component is copied over without any modification, you are recommended to remove the component from the message map. The map transformation will be more efficient since the mapping engine will only focus on the structures that do require change.
- To read elements of a message assembly component, add the component to the input message assembly only. The Mapping node passes it through unchanged
- To modify all the elements of a message assembly component, add the component such as the local environment tree to the input message assembly and to the output message assembly. Then, define transforms between each of its elements.
- To modify some elements of a message assembly component, add the component such as the local environment tree to the input message assembly and to the output message assembly. Define a **Move** transform for the entire component, that is, at folder level, and then specific transforms for each of the elements that you want to transform within an **Override** function. For more information, see "Transforming some elements of a message assembly component by using the Override function" on page 1453.

- To initialize a message assembly component, that is, to create a new message assembly component in your output message, add the message assembly component only to the output message assembly. For more information, see "Initializing a message assembly component in the output message" on page 1458.
- To add a message assembly component, add the message assembly component to the output message assembly and populate at least one field. The Mapping node creates a new output structure containing the results of your transformations.
- To delete a message assembly component from the input message, add the message assembly component to the output message assembly and do not set any field. For more information, see "Deleting a message assembly component from the output message" on page 1457.

Results

The following table summarizes the mapping engine behavior when you add the local environment tree to your message map. The same behavior applies when you add any of the header folders, such as the **MQ headers** > **MQMD** folder, to your message map.

Table 58. Mapping engine behavior when adding the local environment to a message map for transformation

Input element	Output element	Transform defined between the input element and the output element	Mapper behavior
Local environment tree	Local environment tree	A transform operation such as **Move** is defined between the input local environment tree and the output local environment tree. Additional transforms are defined between some elements of the local environment tree within an **Override** function to change the value of those elements.	The input local environment tree is copied into the output local environment tree. The elements whose transforms are defined within the **Override** function have output values different from the input values based on the transformation. The elements outside the **Override** function maintain the same values in the output local environment tree.

Table 58. Mapping engine behavior when adding the local environment to a message map for transformation (continued)

Input element	Output element	Transform defined between the input element and the output element	Mapper behavior
Elements from other message assembly components, database elements whose values are obtained by doing a database read, or a combination of both	Local environment tree	A **Move** transform is defined between each parent input message assembly structure and its corresponding output message assembly structure. Additional transforms are defined between input elements and the output local environment tree. **Note:** If you do not define the **Move** transform between an input and an output message assembly structure from which you use values to set the local environment output elements, then you will lose the message assembly structure in the output message, although your map will perform the transformation correctly.	Each input message assembly structure is copied into its corresponding output message assembly structure. The output local environment variables are defined as per the additional transforms using values from the input elements.
Local environment tree	Local environment tree	None	Delete the original local environment tree. An empty local environment tree is created for the output message.
Local environment tree	None	None	Delete the original local environment tree. An empty local environment tree is created for the output message.
None	Local environment tree	None	Delete the original local environment tree. An empty local environment tree is created for the output message. You might populate some of the fields by using transforms between other input map components such as the message body and the new local environment structure.

Table 58. Mapping engine behavior when adding the local environment to a message map for transformation (continued)

Input element	Output element	Transform defined between the input element and the output element	Mapper behavior
Local environment tree	Local environment tree	A transform operation such as **Move** can be defined between one element from the input local environment tree and one element from the output local environment tree. The rest of the local environment elements do not have transforms defined that specify how to move the input value to the output value.	The output local environment tree is initialized and only the element that has the transform defined has a non default value set.
Local environment tree	Local environment tree	A transform operation such as **Move** is defined between the input local environment tree and the output local environment tree.	The input local environment tree is copied into the output local environment tree.

What to do next

Configure the message map to include the local environment tree or a message header. For more information, see "Customizing a message map to include a message assembly component" on page 1458.

*Transforming some elements of a message assembly component by using the **Override** function*
You can use the **Move** transform to copy a complex type from the input message to the output message, while updating some of the child elements in the complex type by using the **Override** function. A message assembly component is described by a complex data structure.

About this task

Note: You can only use the **Override** function to include **Move** transforms and **Assign** transforms.

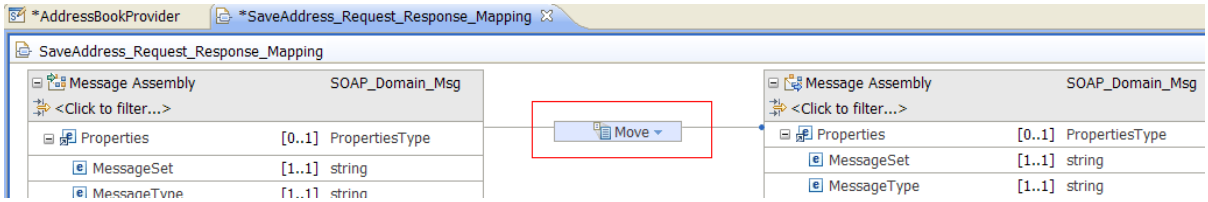
Procedure

When you want to modify only some fields of a message assembly component, complete the following steps to transform the message assembly component:

1. Required: Define a **Move** transform to copy the input message component into the output message component, that is, to copy an input complex data structure into an output complex data structure.

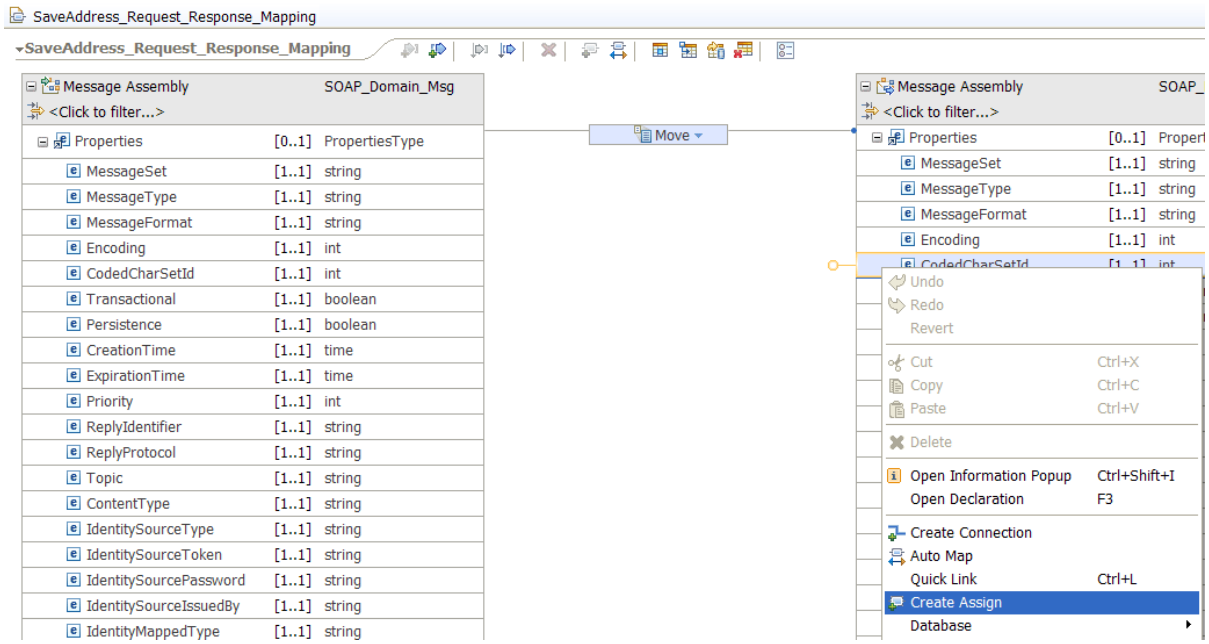
   For example, the Properties tree has a **Move** transform defined automatically when you create a message map so that all elements in the Properties tree are copied to the output Properties tree

structure. If you transform elements in the Local Environment tree, you must manually define the **Move** transform.

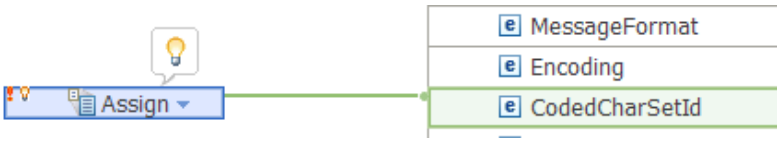


2. Add additional transforms between the input and output elements in the message assembly component.

   For example, you need to change the encoding for the output message. You assign a different value to the **Encoding** element in the properties tree. Right-click the **Encoding** element, and then select the menu option **Create Assign**.

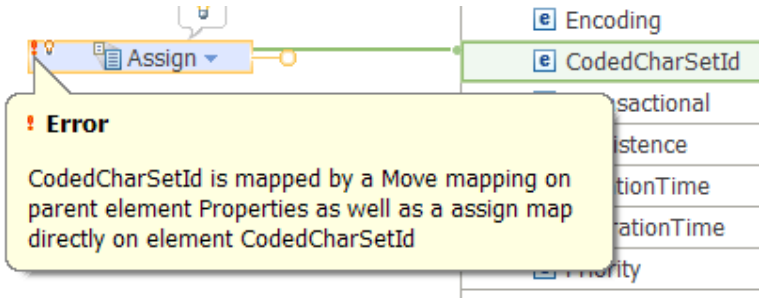


The **Assign** transform is defined and connected to the **Encoding** element in the output Properties tree.



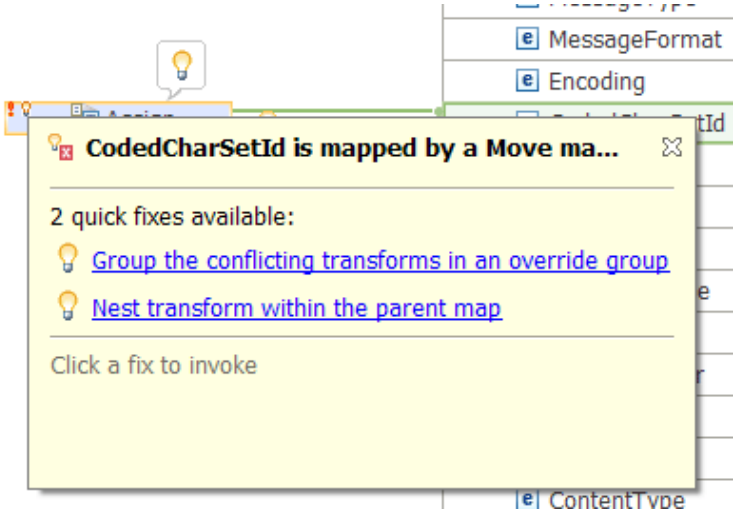
You get the following icons on the top left hand side of the transform:

- An **Error** icon represented with a red exclamation mark. You can ignore this error and continue. You get the error because you have defined two transformations on an element and this is not allowed. By using the **Override** function, you fix the problem.

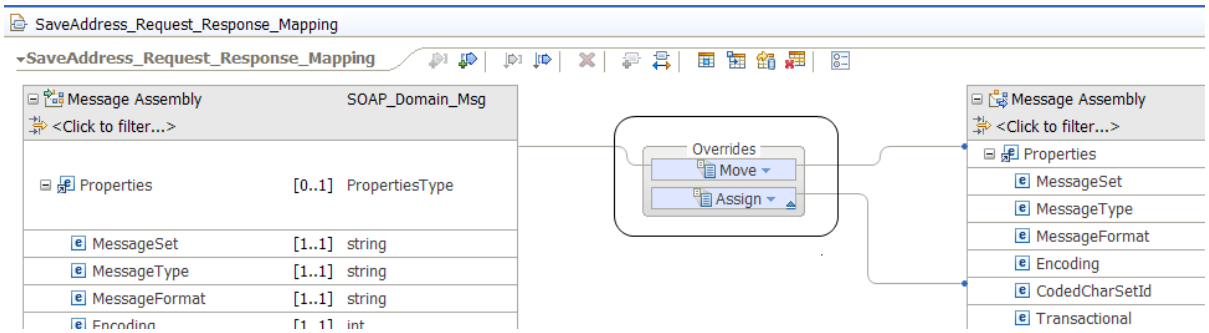


- A **suggestion** icon represented by a yellow light bulb.

  When you hover over the icon, you get the following pop-up window:



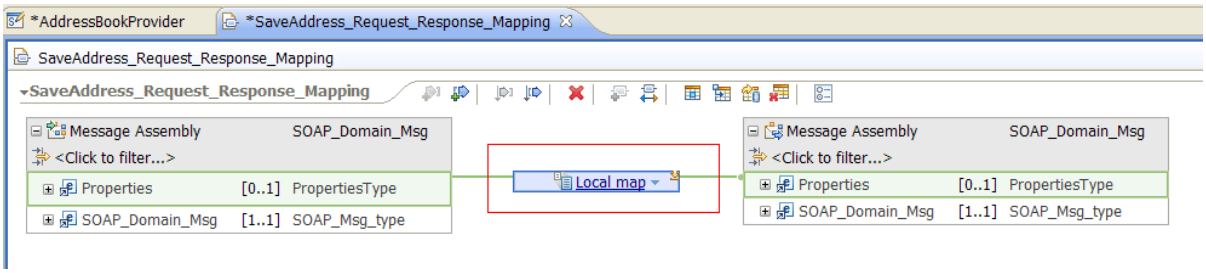
3. When you hover over the yellow light bulb, choose **Group the conflicting transforms in an override group**. This option is the recommended approach and allows you to maintain visibility of the transforms you have defined in the main transformation map.



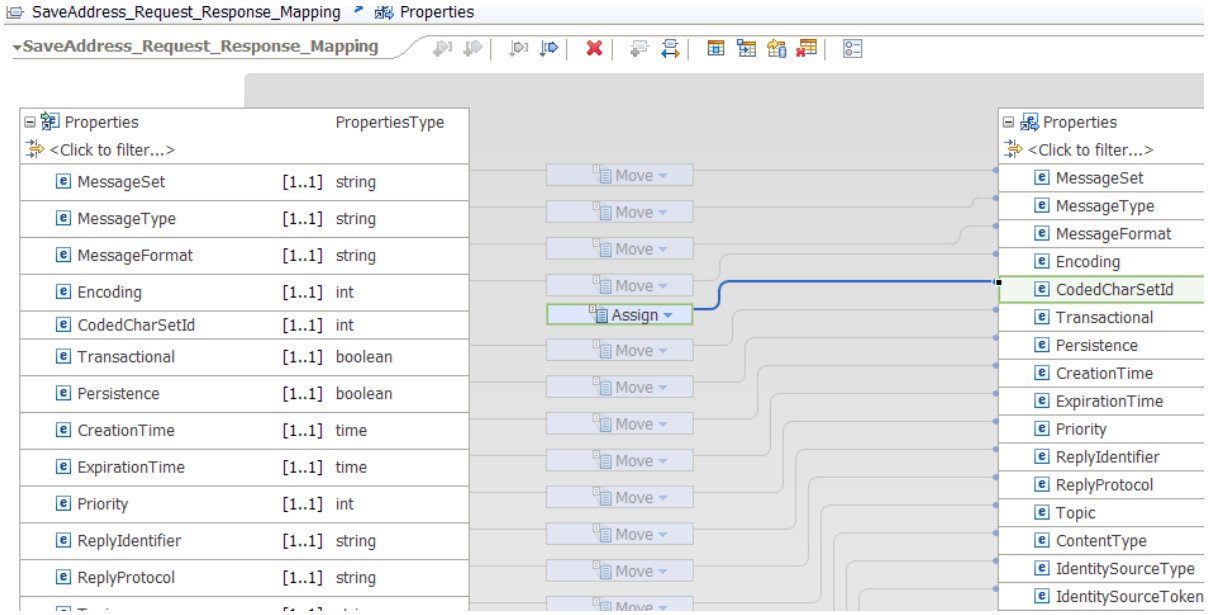
Results

You have transformed elements of a message assembly component by using the **Override** function.

Note: If you choose **Nest transforms within the parent map**, a **Local map** transform is defined between the input Properties tree and the output Properties tree.



The local map that is created contains a **Move** transform per element, with the exception of the **Encoding** element that has an **Assign** transform.



What to do next

Configure the message map body parts. For more information, see “Mapping input to output elements manually” on page 1418 or “Mapping input to output elements automatically” on page 1421.

Deleting a message assembly component from the output message
To delete a message assembly component from the output message, add the message assembly component to the output message and ensure that there are no mappings to it.

About this task

You cannot delete the Environment tree. You can only use the **Remove** transform to explicitly remove elements within the Environment tree.

Procedure

To delete a message assembly component from the output message, complete any of the following steps:

- To delete the local environment tree, add the local environment tree to the output message, and do not set any fields.
- To delete the Properties tree, delete the **Move** transform between the input properties tree and the output properties tree.
- To delete a header, add the transport header folder to the output message, and do not set any fields.
- To delete a message body from the output message, add the message body to the output message, and do not set any fields.

What to do next

Configure the message map body parts. For more information, see “Mapping input to output elements manually” on page 1418 or “Mapping input to output elements automatically” on page 1421.

Initializing a message assembly component in the output message
To initialize a message assembly component in the output message, you add the message assembly component to the output message only. The input values are ignored. You define new values for the elements in the output message assembly structure.

About this task

When you initialize a message assembly structure, you ignore the incoming values and define new values on the output message assembly.

Procedure

To initialize different message assembly components, complete the following steps:

- Add the message assembly component to the output message assembly only.
  - To initialize the local environment tree, add the local environment tree to the output message only.
  - To initialize the Properties tree, delete the **Move** transform between the input properties tree and the output properties tree.
  - To initialize a header, add the transport header folder to the output message only.

  For more information, see "Customizing a message map to include a message assembly component" on page 1458.
- Use mapping transforms to set values for the elements in the new structure.

What to do next

Configure the message map body parts. For more information, see "Mapping input to output elements manually" on page 1418 or "Mapping input to output elements automatically" on page 1421.

Customizing a message map to include a message assembly component
To customize your message map to include more message assembly components, you must add message assembly components to the input message and to the output message, and then define transforms between them.

Before you begin

Check whether you need additional message assembly components to transform your input message. For more information, see "Choosing a mapping action" on page 1450.

About this task

You configure additional message assembly components for a message map in the Graphical Data Mapping editor.

Note: The Environment tree is configured differently. For more information, see "Mapping the environment tree" on page 1471.

By default, when a message map is created, the only message assembly component that is configured automatically is the Properties tree. The input Properties tree is connected to the output Properties tree with a **Move** transform.

Procedure

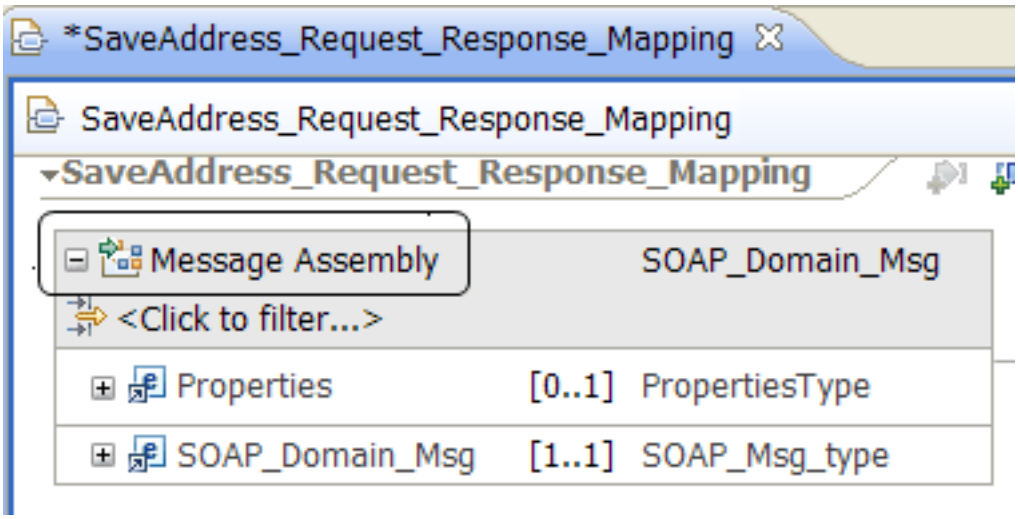
To add a message assembly component in a message map, complete the following steps:

1. Open the message map in the Graphical Data Mapping editor.

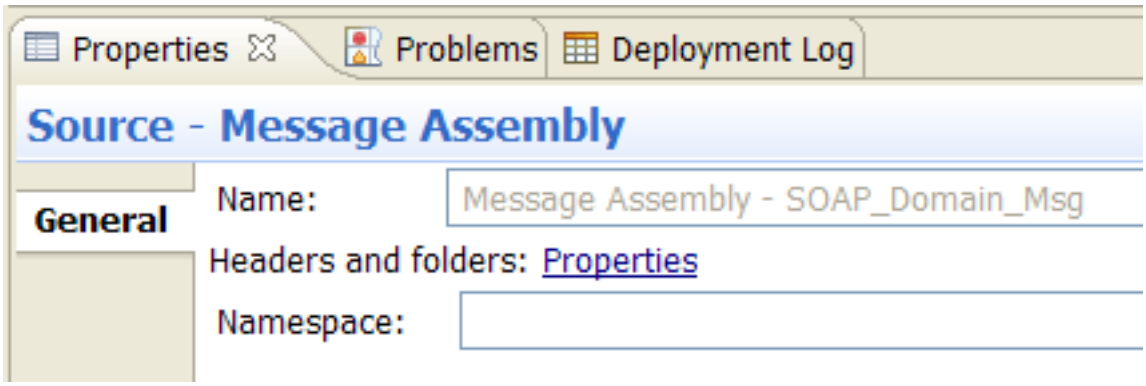
2. Add a message assembly component such as the local environment tree to the input message.
   - Method 1:

     Right-click the input or output message assembly. Then, select **Add or remove headers and folders**.
   - Method 2:

     a. Select **Message Assembly** .



     b. In the Properties view, select the **General** tab.



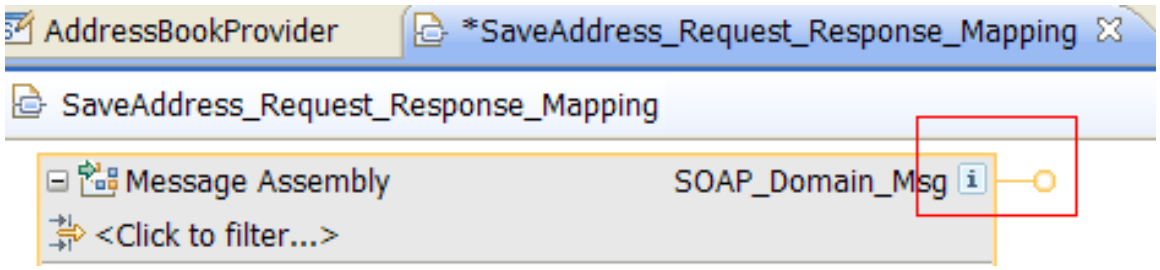
     c. Click **Properties**.

     Note: If you have other structures included in your message assembly, the option that you can click includes all the different message assembly components that you have currently selected.

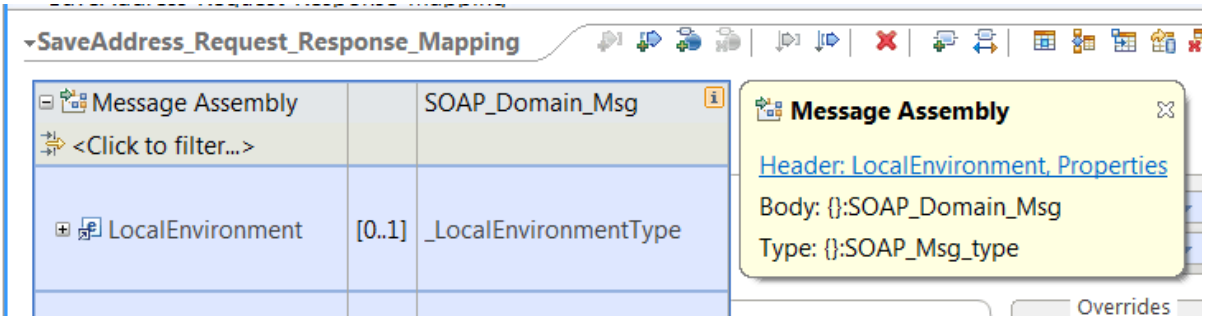
For example, if you had the Properties tree and the local environment tree selected, you click **LocalEnvironment, Properties**.

- Method 3:

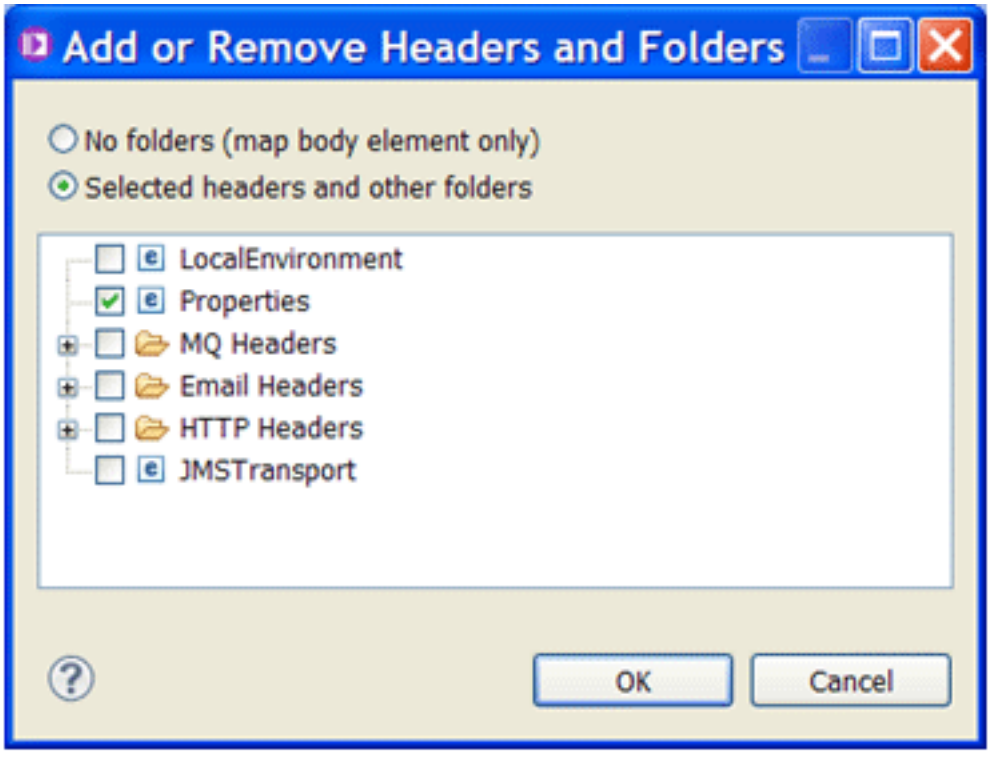
  a. Select the information icon located by the input message body type.



  b. Select **Header: Properties**.



3. In the **Add or Remove Headers and Folders** window, select one or more message assembly components, and then click OK.



What to do next

Define transforms between the input message assembly and the output message assembly. Depending on the data that you want to transform from the added header or folder, it might be necessary to first

define the content of a generic element, such as the Local Environment Variables folder, by using Mapping casts or adding user-defined functions. For more information, see “Mapping data in the local environment tree” on page 1464.

Mapping transport headers
Use the Graphical Data Mapping editor to transform transport headers.

About this task

When you create a top level message map, only the Properties folder is initially included in the map, and a default transformation from input to output properties is created in a local map. You can then use the Message Assembly properties page in the Graphical Data Mapping editor to modify the transport headers that you might include in the map.

Depending on how you add a transport header to a message map, the component can be deleted, initialized, or transformed:

- To pass unchanged a transport header, do not add it to the message map.
- To read elements from a transport header, add it only to the input message assembly of the message map. The Mapping node passes it through unchanged.
- To modify elements in a transport header, add it to input message assembly and to the output message assembly, and provide transforms to copy and modify the header. The Mapping node deletes the input transport header, and creates a new output transport header containing the result of your transformations.
- To add a transport header to your message, add the header to the output message assembly and populate at least one field. The Mapping node creates a new output transport header containing the results of your transformations.
- To delete a transport header, add it to the output message assembly and do not set any field at all. The Mapping node deletes the transport header from the output message.
- MQ Headers

  MQMD
  MQCFH header with root element MQPCF
  MQCIH
  MQDLH
  MQIIH
  MQMDE
  MQRFH
  MQRFH header with MQRFH2 or MQRFH2C parser
  MQRMH
  MQSAPH
  MQWIH
  SMQ_BMH
- Email Headers

  EmailOutputHeader
- HTTP Headers

  HTTPInputHeader
  HTTPReplyHeader
  HTTPRequestHeader
  HTTPResponseHeader
- JMSTransport

Procedure

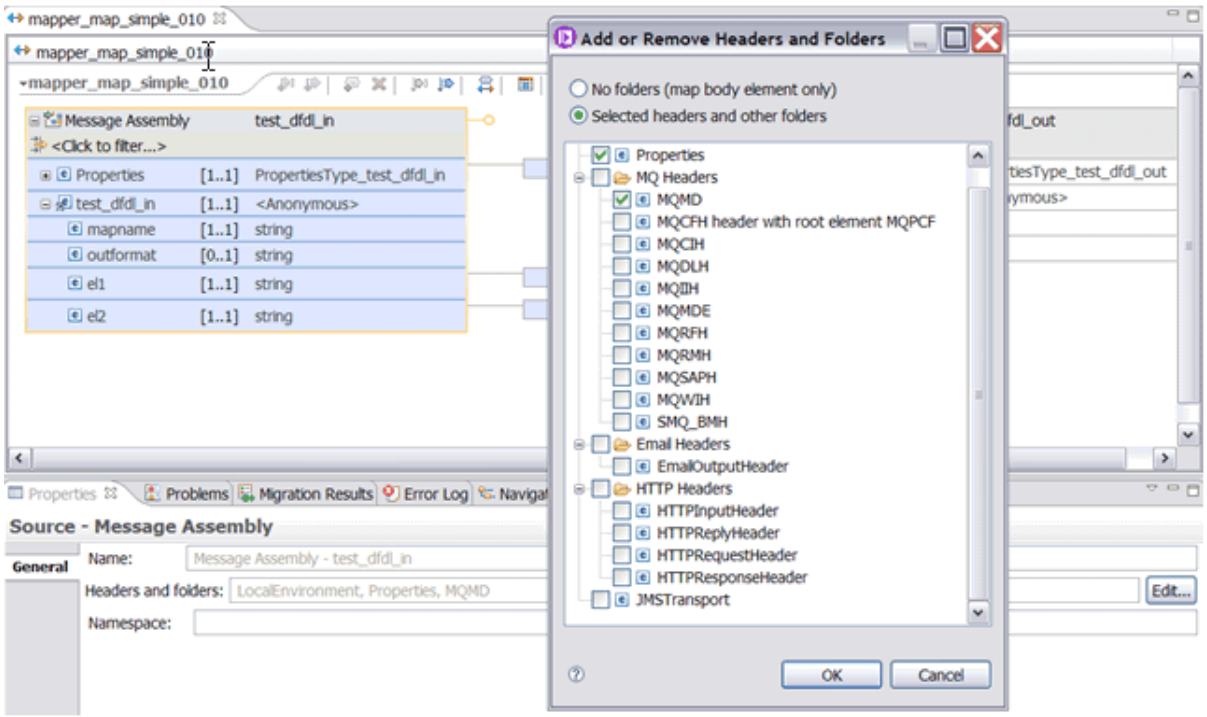
Complete the following steps to transform data in a transport header by using a message map:

1. Decide whether you need a transport header in the input message assembly, the output message assembly, or both. You may want to copy, initialize, or modify elements of a transport header. For more information, see “Choosing a mapping action” on page 1450.

   Note: The Mapping node copies the transport headers from input to output, unchanged, when they are not included in the message map.

2. Add a transport header to the input message assembly, the output message assembly, or both. For more information, see “Customizing a message map to include a message assembly component” on page 1458.

   For example, select **MQMD** to include MQMD headers in the input message assembly of the message map:



3. Optional: Define a **Move** transform between the input transport header and the output transport header to copy all the input values onto the output values.

   Note: If you need to modify only some fields in a transport header, you can use a **Move** transform to copy the transport header unchanged, and then use the **Override** function to modify the elements you must update. For more information, see “Applying mapping overrides” on page 1513.

   You can do this in any of the following ways:

   - In the message map, right-click a transport header on the input message assembly, and select **Add Connection**. Move the mouse to the output local transport header, and click the transport header to define the **Move** transform.
   - In the message map, right-click a transport header on the input message assembly, and select **Quick Link**. A new window appears where you can select a transport header as the output element. Use this option when you have a long list of output elements. You can filter the list in Quick link too.

4. Define transforms between the input transport header and the output transport header. For more information, see “Specifying a transform (mapping operation)” on page 1430 and “Transform types in the Graphical Data Mapping editor” on page 1282.

What to do next

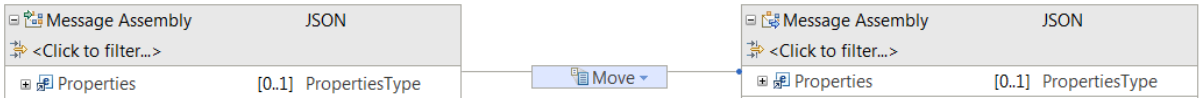
Define additional transformations between other elements in the message map. For more information about the available transforms, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Mapping the Properties tree
In a message map, you can transform elements in the Properties tree by using transforms such as the **Move** transform to copy a value, or the **Assign** transform to set a value of an element. You can also use elements in the Properties tree as input data to your transformations.

About this task

The Properties tree holds information about the characteristics of the message. For more information, see Message tree structure.

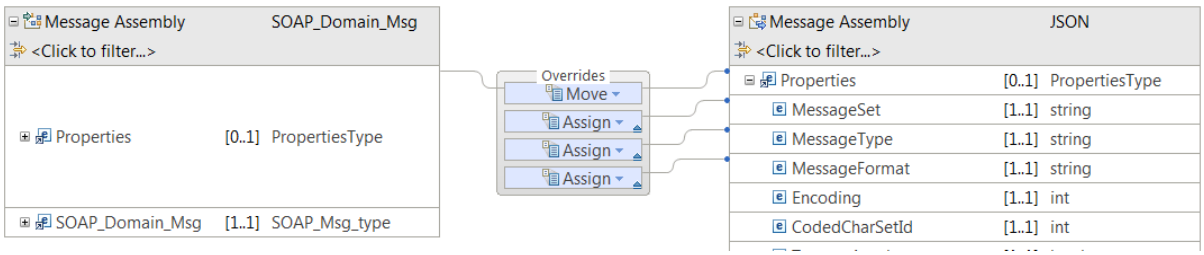
By default, when you create a new map, the Graphical Data Mapping editor includes the Properties tree in the input assembly and in the output message assembly. Depending on the output domain, it can define transforms to update some output elements. Then, copies all the unchanged elements from the input Properties tree to the output Properties tree. If the input and output messages are in the same domain, the transform is a **Move** transform that copies all elements unchanged.



Note: If your mapping does not have any requirements to make changes to the output Properties tree, remove the Properties tree from the output message assembly. The Mapping node will propagate the Properties tree unchanged. If you are also not reading any values from the Properties tree, you can remove the Properties tree from the input message assembly.

When you map messages between unlike domains, it might be necessary to set the **MessageSet**, the **MessageType**, and the **MessageFormat** elements in the output Properties tree. The fields that need setting are dependent on the target parser associated with the output message domain. When you create a new map, the Graphical Data Mapping editor will place a **Move with Overrides** transform group with **Assign** transforms that set default values for these elements in the output.

For example, if the target domain is JSON and the source is SOAP, the Graphical Data Mapping editor defines an **Override** to reset the **MessageSet**, the **MessageType**, and the **MessageFormat**, and move unchanged the rest of the Properties tree elements from input to output.



Note: When you map to a message in the SOAP domain and you have used the **Cast** function to qualify the **xsd:any** in the SOAP Body to a message type defined by a message set format, you must set the **MessageSet**, the **MessageType**, and the **MessageFormat** properties to the values for that message set in the Properties tree.

Procedure

You can complete any of the following mapping tasks when you add the Properties tree to a message map:

1. Use Property elements to define conditional expressions that determine if a transform can be applied anywhere in the map. For more information, see “Defining an XPath conditional expression for a transform” on page 1433.
2. Use the **Assign** transform to set a fixed value without the use of input data.
3. Use transforms to calculate or set the value of an output message element from the value of elements in the Properties tree.

   For example, you can use the **Move** transform to copy an input element value to the output element.
4. Use transforms to calculate the value of a Properties tree element based on the value of elements available in other parts of the message.
5. Use the value of elements defined in the Properties tree to insert, update, or delete data in a database. You can also use the value of a Property element in the definition of a **Where** clause of a **Select** transform.
   - Use the transform **Insert** to add new data to one or more tables in a database.
   - Use the transform **Delete** to delete data in one or more tables in a database.
   - Use the transform **Update** to update data in one or more tables in a database.
   - Use the **Database Routine** transform to update, insert, or delete data in a database by using store procedures.

What to do next

1. Define transforms in the message map to set the value of the output elements. For more information, see “Specifying a transform (mapping operation)” on page 1430.
2. Deploy the map and verify that the output message is valid. For more information, see “Deploying message maps” on page 1585.

Mapping data in the local environment tree
Use the Graphical Data Mapping editor to transform data graphically in the local environment tree.

Before you begin

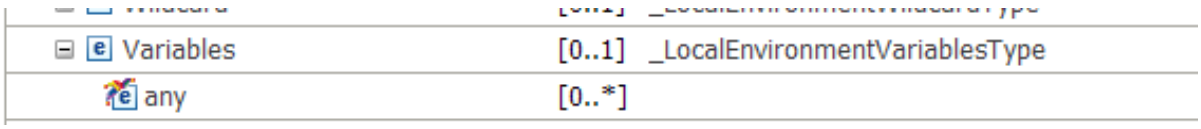
Create a message map. For more information, see “Creating a message map” on page 1382.

About this task

The local environment tree is a part of the logical message tree in which you can store information while the message flow processes the message. You use the local environment tree to store variables that can be referred to and updated by message processing nodes that occur later in the message flow. You can also use the local environment tree to define destinations (that are internal or external to the message flow) to which a message is sent.

When you add the local environment tree to a message map, you must provide transforms for all of its elements so that the input values of each element are not lost. You can copy the input field unchanged or modified by a transform. Many IBM App Connect Enterprise nodes depend on information in the local environment tree being copied along the message flow.

The variables folder in the local environment tree is defined as *xsd:any*. When you add the local environment tree, you can see the structure of the destination folders with all its elements, and a **Variables** folder with a single element defined with a generic type.



You manually define the elements that are included in the **Variables** folder by using the **Cast** or **Add User Defined** functions. There is no predefined structure for the **Variables** folder. Each message flow node has

an input and an output local environment tree. There is a **Variables** folder in the input local environment tree and a **Variables** folder in the output local environment tree.

Note: You should not access attributes or other non-element field types in the local environment tree. The data in the local environment tree is stored by using a generic parser and does not have specific field types set, (see "XMLNSC: Using field types" on page 533). In particular, if you copy data from an XMLNSC tree into the local environment tree, the field type that defines the element or attribute type is not preserved. If you model the data to be an attribute, the map attempts to access the data by using an XPath expression that matches data elements that have the field type that is specified in the XMLNSC tree, and so the data is not read.

When you create a top level message map, only the Properties folder is initially included in the map, and a default transformation from input to output properties is created in a local map.

You can then use the Message Assembly properties page in the Graphical Data Mapping editor to modify the message assembly components transformed in the map and include the local environment folders in the mapping. For more information, see "Customizing a message map to include a message assembly component" on page 1458.

Procedure

Complete the following steps to transform data in the local environment tree by using a message map:

1. Decide whether you need the local environment tree in the input message assembly, the output message assembly, or both. You may want to copy, initialize, or modify LocalEnvironment elements. For more information, see "Choosing a mapping action" on page 1450.

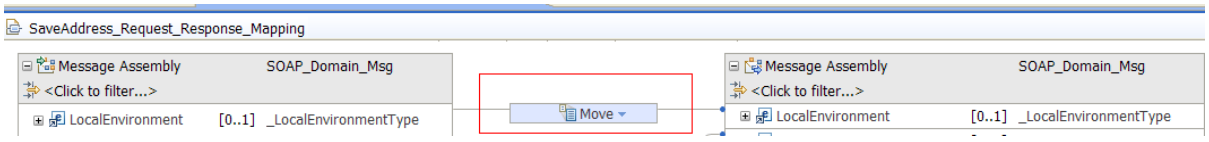
   Note: The Mapping node copies the LocalEnvironment tree from input to output, unchanged, when they are not included in the message map.
2. Add the local environment to the input message assembly, the output message assembly, or both. For more information, see "Customizing a message map to include a message assembly component" on page 1458.
3. Optional: Define a **Move** transform between the input local environment tree and the output local environment tree to copy all the input values onto the output values. Create a connection between the input local environment tree and the output local environment tree.

   Note: If you need to modify only some fields in the local environment tree, you can use a **Move** transform to copy the local environment tree unchanged, and then use the **Override** function to modify the elements you must update. For more information, see "Applying mapping overrides" on page 1513.

   You can do this in any of the following ways:

   - In the message map, right-click **LocalEnvironment** on the input message assembly, and select **Add Connection**. Move the mouse to the output local environment tree, and click **LocalEnvironment** to define the **Move** transform.
   - In the message map, right-click **LocalEnvironment** on the input message assembly, and select **Quick Link**. A new window appears where you can select the output element **LocalEnvironment**. Use this option when you have a long list of output elements. You can filter the list in Quick link too.

   The following figure shows graphically how the **Move** transform is defined between the input local environment tree and the output local environment tree.



4. Optional: Define the **Variables** folder in the local environment tree. The **Variables** folder contains an `xsd:any` element, that you can redefine by using a **Cast** function and a schema file describing the

variables. Alternatively, you can use the **Add User Defined** function to define dynamically the variables that you require in the message map.

- Define the **Variables** folder in the local environment tree by using the **Cast** function. For more information, see "Configuring a generic type in the local environment tree by using the Cast function" on page 1466.
- Define the **Variables** folder in the local environment tree by using the **Add User Defined** function. For more information, see "Configuring a generic type in the local environment tree by using the Add User-Defined function" on page 1470.

5. Define transforms between the input local environment tree and the output local environment tree. For more information, see "Specifying a transform (mapping operation)" on page 1430 and "Transform types in the Graphical Data Mapping editor" on page 1282.

   You must provide transforms to copy or modify all fields that are required on the output. Many message flow nodes depend on information in LocalEnvironment being copied along the message flow. If you need to modify only some fields, use a `Move` transform to copy LocalEnvironment and then use the `Override` function to modify the elements you must update. For more information, see "Applying mapping overrides" on page 1513.

Configuring a generic type in the local environment tree by using the Cast function
You can use the **Cast** function to define variables in a message map that are defined in the local environment tree, such as in the **Variables** folder.

About this task

Sometimes you need to use information that is passed in a generic type variable in the local environment or other part of the message assembly. For example, you might need to calculate the output value of a different element in the message body that is based on a value provided by a preceding node in the local environment variables folder, as shown in the following example:

```
LocalEnvironment.Variables.value_name
```

In this example, `Variables` is a generic element, and you must define the required elements in it for each *value_name*. You can also set values in the output local environment, `Destination.`*`specific_node`*, to override the behavior of a following node. For example:

- For routing in the message flow, the map could set:

  ```
  LocalEnvironment.Destination.RouterList.DestinationData[1].labelName
  ```

- To set a parameter to issue a REST request:

  ```
  LocalEnvironment.Destination.REST.Parameters.parameter_name
  ```

  In this case, `Parameters` is a generic element, and you must define the required elements in it for each *parameter_name*.

- To set an order in a filter, to be used by a LoopBack request:

  ```
  LocalEnvironment.Destination.Loopback.Request.filter.order[1].name
  ```

  In this case, the order is a generic element, and you must define the required elements in it for each *name*.

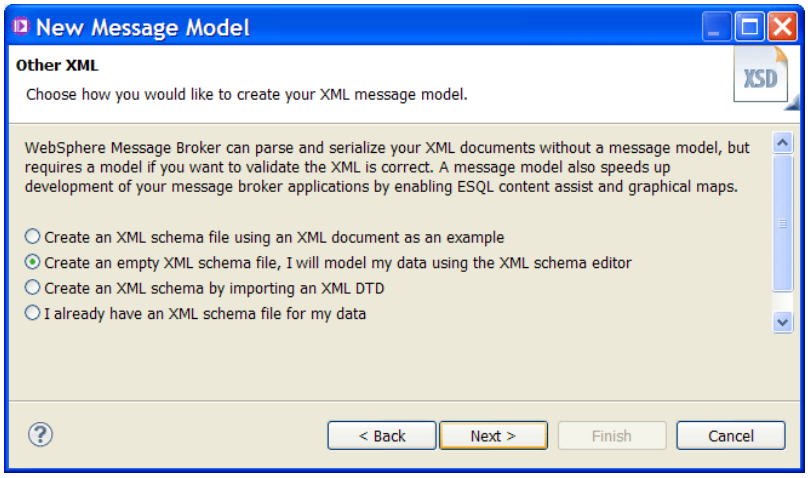
Procedure

To configure the local environment tree **Variables** or other generic element, so that you can use its elements as part of your transformations, complete the following steps:

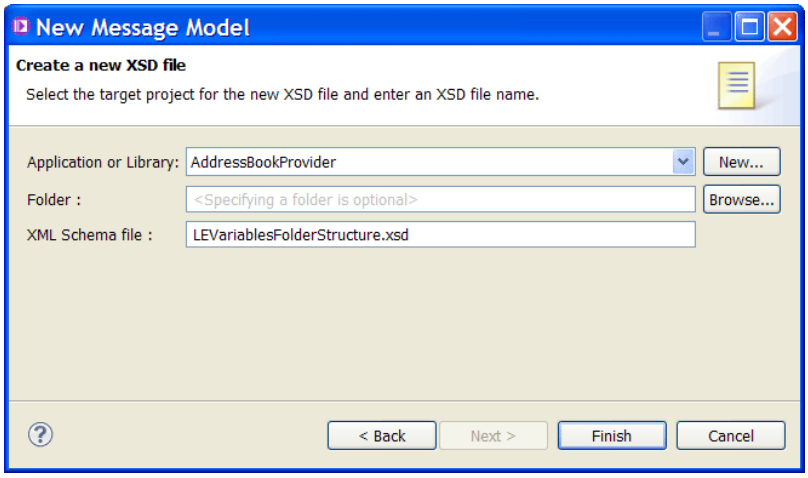
1. If you already have the required element defined in a message model, such as an XML schema, DFDL schema, JSON schema, or Swagger document, omit this step and go to step 2; otherwise, complete this step to create a model to define your data.

   Create a schema file in your application, integration service, or library to define the elements of the local environment tree **Variables** folder and their type:

   - In the **Application Development** view, select **New** > **Message Model** > **Other XML**. Click **Next**.
   - Select **Create an empty XML schema file, I will model my data using the XML schema editor**, and then click **Next**.



   - Create an XSD file ***YourLExsdFileName*.xsd**, where *YourLExsdFileName* is the name of the file that contains the local environment variables folder message model. Then, click **Finish**.



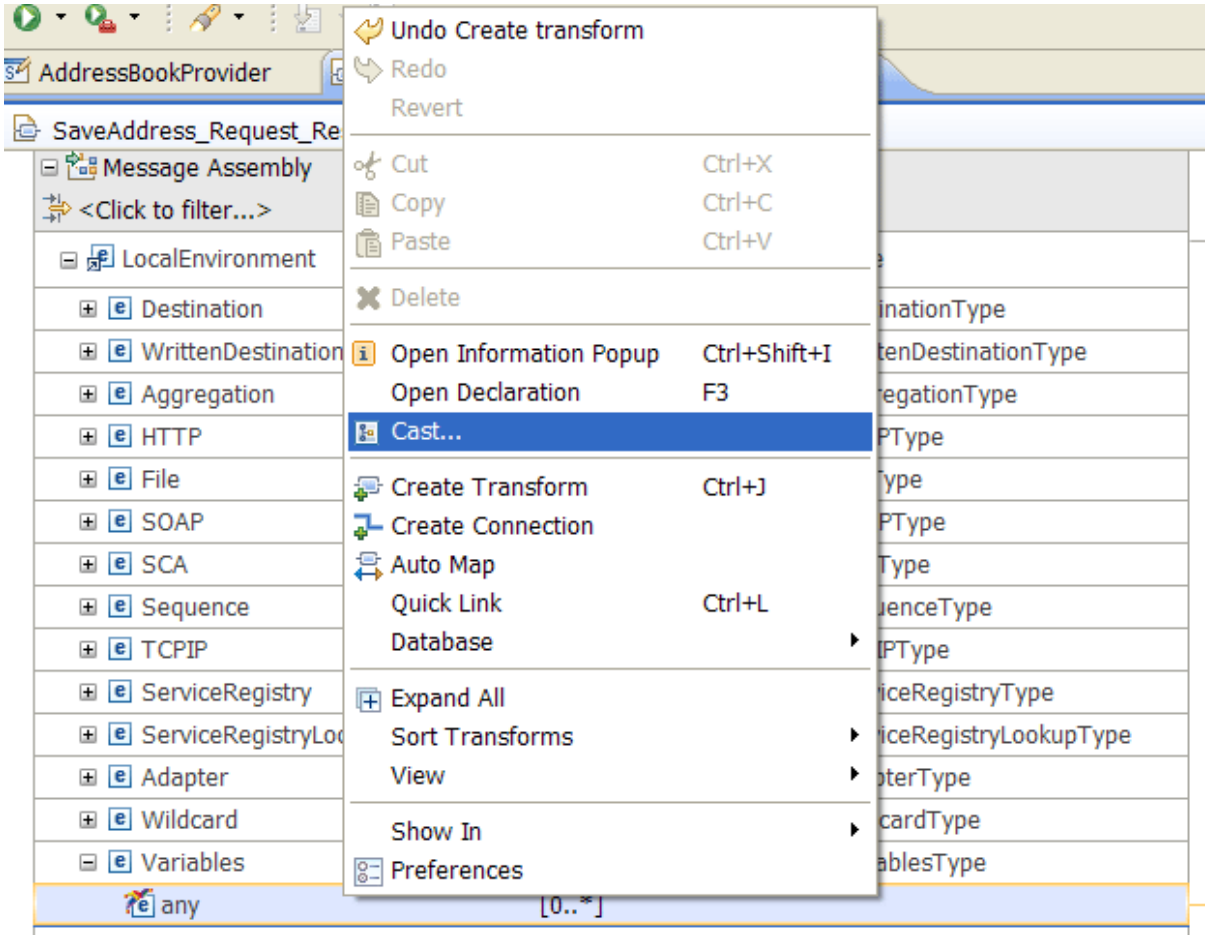
   - The XSD file opens in a new tab where you use the XML Schema editor to define your variables and their types.

     For example, you can define the following schema:

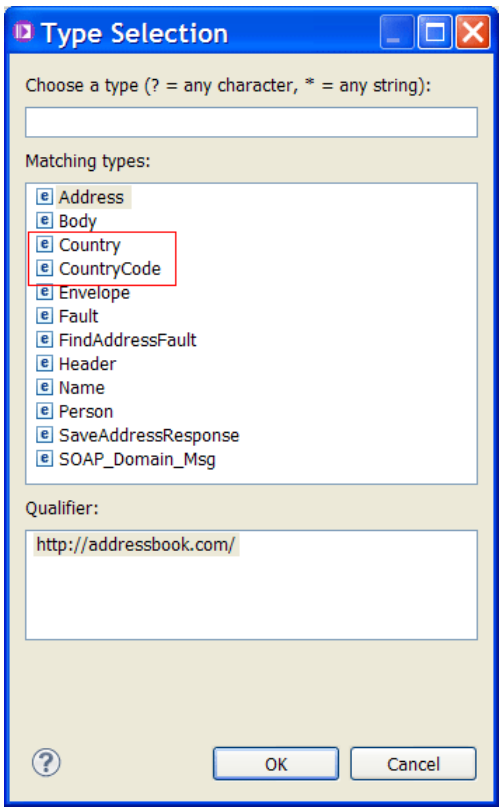
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <xsd:element name="Country"  type="xsd:string"/>
       <xsd:element name="CountryCode"  type="xsd:integer"/>
</xsd:schema>
```

     Note: You can define the local environment variables in a message flow node by using ESQL or Java. Namespaces is not defined. For this reason, the schema is also defined without a namespace declaration.

2. Use the **Cast** function to define the local environment variables in the message map so they are visible under the **Variables** folder in the map. Complete the following steps to cast the **any** element to a variable and its type in the output local environment tree:
   - Right-click the **any** element, and then select **Cast**.



   - In the **Type Selection** window, select a type, for example **Country**, and then click **OK**.



Note: Do not cast to an XML type that uses attributes. The data in the local environment tree is stored by using a generic parser and does not have specific field types set, (see "XMLNSC: Using field types" on page 533). In particular, if you copy data from an XMLNSC tree into the local environment tree, the field type that defines the element or attribute type is not preserved. If you model the data to be an attribute, the map attempts to access the data by using an XPath expression that matches data elements that have the field type that is specified in the XMLNSC tree, and so the data is not read.

Results

You have defined one local environment variable that can be used by other message flow nodes in your message flow for routing or filtering.

You can see the element **Country** under the local environment **Variables** folder in the message map.

Variables	[0..1]	_LocalEnvironmentVariablesType
any	[0..*]	
Country	[0..*]	string

Example

Another example:

If you set in an ESQL compute node two simple fields within the **Variables** folder of the Local Environment tree by using the following code:

```
SET Outputlocal environment.Variables.dec = 10.1;
SET Outputlocal environment.Variables.str = 'Some text';
```

To access these fields in a Mapping node by using the **Cast** function, you must create a schema file in your integration solution to define the elements and their type. Note that since the ESQL is not using any namespace to qualify these elements, the schema is also defined without a namespace declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <xsd:element name="dec"  type="xsd:decimal"/>
       <xsd:element name="str"  type="xsd:string"/>
</xsd:schema>
```

Once the schema file is saved, you can then select the any element under the Variables section in the Local Environment tree, and use the context menu action **Cast** to add a Mapping cast for each of the elements "dec" and "str" that are required in the message map. For more information, see "Mapping xsd:any on an input or output message" on page 1397.

What to do next

Define transforms between the input local environment tree and the output local environment tree. For more information, see "Specifying a transform (mapping operation)" on page 1430 and "Transform types in the Graphical Data Mapping editor" on page 1282.

Configuring a generic type in the local environment tree by using the Add User-Defined function
You can use the **Add User-Defined** function to define variables in a message map that are defined in the local environment tree, such as in the **Variables** folder.

About this task

Sometimes you need to use information that is passed in a generic type variable in the local environment or other part of the message assembly. For example, you might need to calculate the output value of a different element in the message body that is based on a value provided by a preceding node in the local environment variables folder, as shown in the following example:

```
LocalEnvironment.Variables.value_name
```

In this example, `Variables` is a generic element, and you must define the required elements in it for each *value_name*. You can also set values in the output local environment, `Destination.`*`specific_node`*, to override the behavior of a following node. For example:

- For routing in the message flow, the map could set:

  ```
  LocalEnvironment.Destination.RouterList.DestinationData[1].labelName
  ```

- To set a parameter to issue a REST request:

  ```
  LocalEnvironment.Destination.REST.Parameters.parameter_name
  ```

  In this case, `Parameters` is a generic element, and you must define the required elements in it for each *parameter_name*.

- To set an order in a filter, to be used by a LoopBack request:

  ```
  LocalEnvironment.Destination.Loopback.Request.filter.order[1].name
  ```

  In this case, the order is a generic element, and you must define the required elements in it for each *name*.

Procedure

To configure the local environment tree **Variables** or other generic element, so that you can use its elements as part of your transformations, complete the following steps:

1. Identify the variables that you must add in your message map under the **Variables** folder.

2. Use the **Add User-Defined** function to define one by one each variable in the message map. For more information, see “Adding and renaming a user-defined element” on page 1410.
3. Select the type for each user-defined variable. For more information, see “Defining the structure of a complex user-defined element” on page 1414.

Results

You have a message assembly structure that includes the local environment tree, and its **Variables** folder includes an any element, and other elements.

What to do next

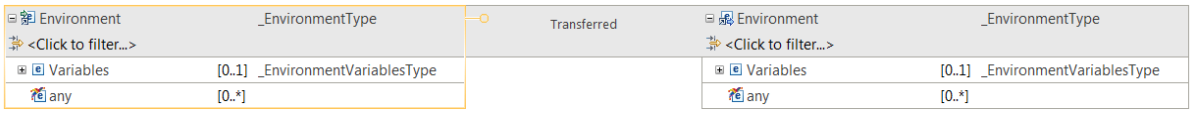
Define transforms between the input local environment tree and the output local environment tree. For more information, see “Specifying a transform (mapping operation)” on page 1430 and “Transform types in the Graphical Data Mapping editor” on page 1282.

Mapping the environment tree
In a message map, you can update, delete, or create data in the environment tree **Variables** folder. You can use the environment tree as input data to your transformations.

About this task

The environment tree is a single global set of data that is available to all nodes in a message flow. The Mapping node accesses the single instance of the environment tree at both the input and output side of the map.

The Graphical Data Mapping editor displays the environment tree by showing the same component on the input and in the output connected via a banner to highlight the fact that it is a single global object.



In the Graphical Data Mapping editor, you can add the environment tree to the input message assembly of a message map or to the output message assembly of a message map.

- When you add the environment tree to the input message assembly, you automatically get the environment tree included in the output message assembly.
- When you add the environment tree to the output message assembly, you automatically get the environment tree included in the input message assembly.
- When you define the content of the environment tree, changes are automatically applied to both the input and the output message assembly.

To add the environment tree, select the icon  in the Graphical Data Mapping editor toolbar.

Alternatively, you can right-click anywhere in the map and select **Add environment mapping**.

To remove the environment tree, select the icon  in the Graphical Data Mapping editor toolbar. Alternatively, you can right-click anywhere in the map and select **Remove environment mapping**.

Note: When you use the environment tree in your integration solutions, you should create your own information in the provided folder called **Variables**.

You define the structure of the environment tree by using the **Add User-Defined** function or the **Cast** function. For more information, see “Adding and renaming a user-defined element” on page 1410 or “Casting elements in a message map” on page 1398.

The following rules apply when you map the environment tree in the Graphical Data Mapping editor:

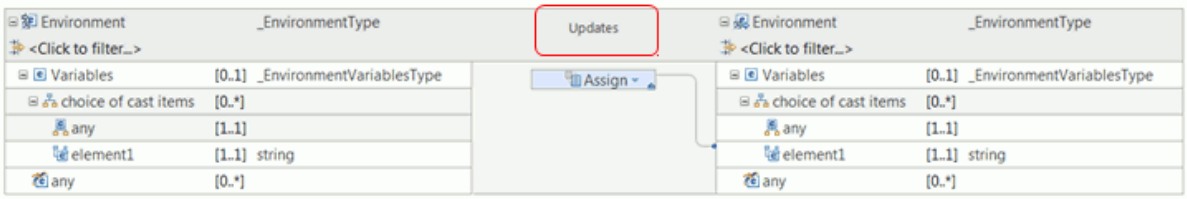
- If you do not include the environment tree in your message map, the entire contents of the environment tree are retained unchanged.
- If you only need to read and update the environment tree, you define a map that only contains the Environment tree.
- If you only need to read from the input assembly and update the environment tree, you define a map that includes the input message assembly and the environment tree.
- If you only need to read from the environment tree and build a new output message, you define a map that includes the output message assembly and the environment tree. You can optionally update the environment tree.
- If you have multiple output message assemblies, or the output message assembly is propagated multiple times, for example when you split an input message with a repeating structure into multiple messages, the environment tree will only be mapped once.
- You cannot wire an output message assembly and the environment tree to a transform that allows multiple outputs, that is, an **If** transform, a **For each** transform, and any of the database transforms. When you need to populate both Environment and the output message assembly, you must have two instances of the transform.

When you add for the first time the environment tree in your message map, the entire contents of the environment tree are retained unchanged. This is indicated visually in the banner with the word **Transferred** located between the input and the output environment structures.

If you include the environment tree in your message map, then the entire contents of the input environment tree are retained in the output environment tree, subject to any modifications that you make in the map.

- Any element present in the environment tree that is not transformed in the map remains unmodified.

  Note: Yon only need to define the elements that you need to map to or from the environment tree. All other elements that might be present in environment tree will remain and be transferred unmodified by the map. You can use the **Remove** transform to delete data from the environment tree.
- When you add transformations that modify the environment tree, there will be a visual change to indicate that the environment tree will be updated after being transferred. As soon as you modify an element in the environment tree, the word **Transferred** changes to **Updates**.



- When you define a transform to modify an element in the environment tree, the value of the input element is not transferred. The outcome of the transform determines the existence, state, and content of the element in the environment tree output element.

Note: For each element in the environment tree, you cannot have different transforms that use the element as a both a source and a target.

Procedure

You can complete any of the following mapping tasks when you add the environment tree to a message map:

1. Use Environment elements to define conditional expressions that determine if a transform can be applied. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.
2. Use the value of elements defined in the **Variables** folder of the environment tree to calculate or set the value of an output message element. Use the value of elements from other parts of the message to calculate the value of an environment tree element.
   - Use the **Assign** transform to set a fixed value without the use of input data.
   - Use the **Create** transform to create an empty element, or a nil element without the use of input data.
   - Use the **Move** transform to copy an input element value to the output element.
   - Use the **If** transform, a custom transform, or the **XPath** transform to define the condition that determines the value of an output element.
   - Use the **Append** transform, the **Group** transform, the **ForEach** transform, the **Join** transform, a custom transform, or the **XPath** transform to set the value of a repeating element.
   - Use the **Select** transform or the **Database Routine** transform to set the value with data available in a database system.
   - Use the **xs:type** transforms to cast the value of a simple input element to a specific data type.
   - Use the **fn:type** transforms to set the value by using XPATH 2.0 functions.

   In the same map, you cannot access and update an element in the environment tree. For example, you cannot define a **Move** transform and an **Assign** transform to the same environment element in a map. Maps do not allow you to control the order of transforms. In this example, you cannot control whether the access or the update of the element occurred first. You will get an error if you access and update an element in the environment tree in the same map.

   Note: You can define a custom transform, **Custom XPath**, **Custom Java**, or **Custom ESQL**, to read the value of an environment element, and then increment its value. The Graphical Data Mapping editor reads the environment variable first, and then updates its value. For example, you can use it to implement a counter action.

   The following transforms are not permitted in environment tree mappings:

   - **Local Map** transform.
   - **Submap** transform.

   When you use a **Custom ESQL** transform or a **Custom Java** transform to modify the environment tree in a message map, check that the transform does not make modifications to environment data that might conflict with other transforms in the map.

   If you use a mapping to add a child to a complex element which already exists in the environment tree, it will be added as the last child.

   If you wish to modify a child element of an existing repeating parent, you should iterate over each parent using the **ForEach** transform, and include child mappings.

   If you need to read and update an element of the Environment tree, you must use two sequential Mapping nodes.

3. Use the value of elements defined in the **Variables** folder of the environment tree to insert, update, or delete data in a database. You can also use the value of an environment element in the definition of a **Where** clause of a **Select** transform.
   - Use the transform **Insert** to add new data to one or more tables in a database.
   - Use the transform **Delete** to delete data in one or more tables in a database.
   - Use the transform **Update** to update data in one or more tables in a database.
   - Use the **Database Routine** transform to update, insert, or delete data in a database by using stored procedures.

4. Use the **Remove** transform to remove elements from the environment tree.

   You do not need to define a **Move** transform between the input and the output environment structures, data will be retained. If you want to remove data from the environment tree, you can use the **Remove** transform. For more information, see "Remove" on page 1336.

What to do next

Define transforms in the message map to set the value of the output elements. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Adding database definitions to the IBM App Connect Enterprise Toolkit
You must have a database definition (`.dbm` file), contained in a data design project, to create database mappings by using the Mapping node.

About this task

A database definition file holds the physical data model that details all the database resources, such as the schema, the tables, and other resources, that you need access to.

If you can connect to your database server by using the IBM App Connect Enterprise Toolkit, you can create a database definition when you create your database mapping. For more information, see "Mapping database content" on page 1523.

If you cannot connect to your database server by using the IBM App Connect Enterprise Toolkit, you must create a database definition file from scratch before you can create your database mapping. For more information, see "Creating a database definition from scratch" on page 1478.

You can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables. You must include a data design project in an application, or reference it from an Integration project, before you can use the database definitions that the data design project contains.

Database definition files in the IBM App Connect Enterprise Toolkit are not automatically updated. If you modify your database, you must re-create the database definition files.

The following topics describe how to add a database definition to the IBM App Connect Enterprise Toolkit:

- "Creating a data design project" on page 1474
- Creating a database definition by using the **New Database Definition File** wizard to connect to a database server.
- Create a database definition from scratch when it is not possible to connect to the database server by using the IBM App Connect Enterprise Toolkit.
- "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475
- "Creating a database definition from scratch" on page 1478

Creating a data design project
Create a data design project to contain your database definition files.

About this task

A data design project is a specialized type of project where you store database definition files that hold information about database resources.

To create a data design project in the Integration Development perspective, complete the following steps:

Procedure

1. Click **File** > **New** > **Other**.
   A window opens in which you can select a wizard.
2. Expand **Data**, select **Data Design Project**, and click **Next**.
   The **New Data Design Project** wizard opens.
3. Enter a name for your data design project, and then click **Finish**.
   The **Open associated perspective** dialog is displayed.
4. Click **No**.
   Your data design project is created, and is displayed in the **Application Development** view, under **Independent resources**.

What to do next

After you create a data design project, you can complete the following tasks:

- Create a database definition by using the **New Database Definition File** wizard to connect to a database server; see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.
- Create a database definition from scratch when it is not possible to connect to the database server by using the IBM App Connect Enterprise Toolkit; see "Creating a database definition from scratch" on page 1478.
- Include your data design project in an application; see Adding a project to an application, integration project, or library.
- Reference your data design project from an Integration project; see Referencing resources in other libraries.

*Creating a database definition (`.dbm` file) by using the **New Database Definition File** wizard*

Use the **New Database Definition File** wizard to add database definitions to the IBM App Connect Enterprise Toolkit.

Before you begin

Note: Your database server must be configured to listen on a static port; dynamic ports are not supported with the **New Database Definition File** wizard. Contact your database administrator for information on how to verify port configuration.

About this task

A database definition file holds the physical data model that details all the database resources, such as the schema, the tables, and other resources, that you need access to.

If you can connect to your database server by using the IBM App Connect Enterprise Toolkit, you can create a database definition (`.dbm` file) when you create your database mapping. For more information, see "Mapping database content" on page 1523.

If you cannot connect to your database server by using the IBM App Connect Enterprise Toolkit, you must create a database definition file from scratch before you can create your database mapping. For more information, see "Creating a database definition from scratch" on page 1478.

You can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables. You must include a data design project in an application, or reference it from an Integration project, before you can use the database definitions that your data design project contains.

Database definition files in the IBM App Connect Enterprise Toolkit are not automatically updated. If you modify your database, you must re-create the database definition files.

Complete the following steps to create a database definition (`.dbm` file) by using the **New Database Definition File** wizard:

Procedure

1. Click **File** > **New** > **Database Definition**.

   The **New Database Definition File** wizard is displayed.
2. Select an existing data design project, or click **New** to create a data design project.
3. From the **Database** drop-down list, select the type of database that you want to model and, if applicable, select the **Version**. Click **Next**.

   Important: Ensure that you select a database from the list that is supported by IBM App Connect Enterprise; you can use this wizard in a shell-share environment with other Rational products that support other databases or versions. For a list of databases supported by IBM App Connect Enterprise, see IBM App Connect Enterprise requirements.

   If your database is supported by IBM App Connect Enterprise, but is not included in the list of selectable databases, you might need to create a database definition from scratch; see "Creating a database definition from scratch" on page 1478.

   When using specific database servers, you might need to set the paths of your JDBC JAR files before you can use them with the **New Database Definition File** wizard. To set the paths of your JDBC JAR files:

   a) Click **Window** > **Preferences**.

      The **Preferences** window opens.

   b) Expand **Data ManagementConnectivityDriver Definitions**.

   c) In the **Driver Definitions** pane, select the database server that you want to connect to, and click **Edit...**.

      The **Edit Driver Definition** window opens.

   d) Select the **JAR List** tab, and click **Add JAR/Zip...**.

   e) Browse to the JDBC JAR file that was supplied with your database product, select the JAR file, and then click **Open**.

      The **Edit Driver Definition** window closes.

   f) In the **Preferences** window, click **OK** to close the window, and return to the **New Database Definition File** wizard.
4. Select a connection to use from the list of existing connections, or click **New** to create a database connection.

   If you select to use an existing connection, the existing database definition is overwritten.

5. If you selected to create a connection:
   a) Optional: If you want to enter a custom value for the *Connection Name*, clear **Use default naming convention**.
   b) Enter values for the Connection to the database, for example, *Database name*, *Host name* and *Port number*.
   c) Enter values for the *User ID* and *Password* to connect to the database.

      Click **Test Connection** to verify the settings that you selected for your database. The default Port number for a DB2 database is 50000.
   d) Click **Next**.

      If your data design project already contains a database definition for the database that you selected, and you want to overwrite this database definition, click **Yes** in the **Confirm file overwrite** window.

      Tip: If you cannot connect to your database by using the **New Database Definition File**, you might need to create you database definition from scratch; see "Creating a database definition from scratch" on page 1478.
6. Alternatively, if you selected to use an existing connection:
   a) To overwrite the existing database definition, click **Yes** in the **Confirm file overwrite** dialog.
   b) Enter values for the *User ID* and *Password* to connect to the database, and then click **Next**.
7. Select one or more database schemas from the list and click **Next**.

   When you create a database definition for use in a graphical data map called from a Mapping node, the database schema name that is displayed in the map is the one that you select here, but might be overridden by the corresponding JDBC Providers policy.
8. Select the elements that you require on the **Database Elements** page.

   You can select any option, in addition to **Tables**, on the **Database Elements** page.
   a) Select **Views** to see all the database views in the **Data Project Explorer**
   b) Select **Routines** to add stored procedures and user-defined functions to the database definition file.

   If you select additional options, the database definition files that you create contain more information than the Compute, Database, or Mapping nodes require.
9. Click **Finish**.

Results

A new database definition (`.dbm` file) is added to your data design project. If you opened the **New Database Definition File** wizard while creating a database transform in the graphical data mapping editor, you are returned to the database transform wizard.

What to do next

Before you can use your database definition in a messaging solution, you must complete one of the following tasks:

- Include your data design project in an application; see Adding a project to an application, integration service, or library.
- Reference your data design project from an Integration project; see Referencing resources in other libraries.
- Create a corresponding JDBC Providers policy; see JDBC Providers policy (JDBCProviders).

Creating a database definition from scratch
You can create a database definition (`.dbm` file) from scratch. A database definition is required to create database mappings.

Before you begin

- You must have created a data design project; see "Creating a data design project" on page 1474

About this task

A database definition file holds the physical data model that details all the database resources, such as the schema, the tables, and other resources, that you need access to.

You must have a database definition (`.dbm` file) contained in a data design project before you can create database mappings. You can also use database definitions in other nodes, such as the Compute node, to validate references to database sources and tables. You must include a data design project in an application, or reference it from an Integration project, before you can use the database definitions that your data design project contains.

Database definition files in the IBM App Connect Enterprise Toolkit are not automatically updated. If you modify your database, you must re-create the database definition files.

Complete the following steps to create a database definition (`.dbm` file) from scratch:

Procedure

1. Click **File** > **New** > **Other**.
   A window opens in which you can select a wizard.
2. Expand **Data**, select **Physical Data Model**, and click **Next**.
   The **New Physical Data Model** wizard opens.
3. Next to the *Destination folder* field, click **Browse...**.
   A window opens in which you can select a parent folder for your database definition.
4. Select a data design project from the list, and then click **OK**.

   Important: Ensure that you select a data design project as the parent folder for your database definition. Database definitions must be contained in a data design project before you can use them in your IBM App Connect Enterprise messaging solutions.
5. In the *File name* field, enter a name to represent the database that you want to model. You do not need to select the **Database** or **Version**.

   Important: Create a database definition only if your database is supported by IBM App Connect Enterprise.

   For a list of databases supported by IBM App Connect Enterprise, see IBM App Connect Enterprise requirements.
6. Select **Create from template**, then click **Next**.
7. In the **Templates** pane, select **Empty Physical Data Model**, then click **Finish**.
   Your empty database definition is created, is displayed in the Data Project Explorer view, and is opened in the **Physical Data Model** editor.
8. If the Data Project Explorer view is not open in your IBM App Connect Enterprise Toolkit, open it:
   a) Click **Window** > **Show View** > **Other**.
      A window opens in which you can select a view.
   b) Expand **Data Management**, select **Data Project Explorer**, and click **OK**.
      The **Data Project Explorer** view opens.
9. In the Data Project Explorer view, expand your database definition and select **Database**.
   The database properties are displayed in the Properties view.

10. In the Properties view, select the **General** tab. In the *Name* field, enter the name of your database.

    If you use this database definition with the Graphical Data Mapping editor, the *Name* is displayed as the name of the data source, and is used when creating the JDBC Providers policy for the IBM App Connect Enterprise runtime connection.
11. In the Data Project Explorer view, select **Schema**.

    The database schema properties are displayed in the Properties view.
12. In the Properties view, select the **General** tab. In the *Name* field, enter the name of your database schema.

    Database schemas are used only by the Mapping node, and only when calling a graphical data map that contains a database transform. For more information about mapping database content, see "Mapping database content" on page 1523.
13. In the Data Project Explorer view, right-click **Schema** and select **Add Data Object** > **Table**.

    A **Table** is created, and is displayed under your **Schema** in the Data Project Explorer view.
14. Define the columns in your **Table**:
    a) In the Data Project Explorer view, right-click your **Table**, and select **Add Data Object** > **Column**.

       A **Column** is created, and is displayed under your **Table** in the Data Project Explorer view.
    b) Enter a name for your **Column**.
    c) In the Properties view, select the **Type** tab to define the attributes of your column.
15. Repeat steps "13" on page 1479 through "14" on page 1479 for each table in your database, and then save your database definition.
16. Save your database definition, and close the **Physical Data Model** editor.

What to do next

Before you can use your database definition in a messaging solution, you must complete one of the following tasks:

- Include your data design project in an application; see Adding a project to an application, integration service, or library.
- Reference your data design project from an Integration project; see Referencing resources in other libraries.

Accessing user-defined properties from a Mapping node

You can use a Mapping node to access properties that are associated with the message flow that contains the node.

About this task

To access these properties from a Mapping node, call the function `iib:getUserDefinedProperty('propertyname')` from a **Custom XPath** transform. The function returns a string that contains the property value, regardless of the original type of the property.

Note: In IBM Integration Bus Version 10, the function `mb:getUserDefinedProperty('propertyname')` is deprecated. If you try to use this function, you get the following message: `The function mb:getUserDefinedProperty in XPath mb:getUserDefinedProperty('UDP1') is deprecated. Use the corresponding "iib:" function instead.`

Procedure

To access user-defined properties from a Mapping node, complete the following steps:

1. Right-click the element for which you want to set the value by using a user-defined property, and click **Add Assign**.
2. Click the down arrow on the transform and change the **Assign** transform to a **Custom XPath** transform.

3. In the **General** tab of the **Properties** page for the **Custom XPath** transform, enter the following text:

```
iib:getUserDefinedProperty('property_name')
```

where *property_name* is the name of your user-defined property.

Results

You set the value of your element to a user-defined property. Now you can provide a value for the user-defined property dynamically by using the Integration API or a BAR file override.

Setting the value of an output element by using a transform or a function

Use the Graphical Data Mapping editor to set the value of an output element by using an expression, a transform, or a function.

About this task

You can use a function or a transform to set the value of an output element, either by connecting the output element with an input element and then specifying a transform on the connection between them, or by specifying a transform directly on the output element. For information about creating connections and specifying transforms, see "Editing message maps" on page 1394.

For information about all the functions and transforms that are available, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Transforms that support conditional control such as the **If** transform can use XPath 2.0 expressions, or invoke Java or ESQL functions.

Procedure

You can use any of the following mapping operations to map graphically your data in the Graphical Data Mapping editor

- **Core mapping transforms**: You can use built-in structural and functional mapping operations, which enable you to graphically construct the required message transformations to build the output message. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.
- **Custom transforms**: You can use custom transformations to build your own XPath 2.0, Java, or ESQL functions, which can be invoked to perform specialized transformations within the message map. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.
  - Custom Java transform. For more information, see "Custom Java" on page 1312.
  - Custom XPath transform. For more information, see "Custom XPath" on page 1315.
  - Custom ESQL transform. For more information, see "Custom ESQL" on page 1308.
- **XPath functions** (`fn:`*`functionName`*): You can use XPath 1.0 and XPath 2.0 functions to transform data in a message map. For more information about XPath, see the online document W3C XML Path Language (XPath) 2.0.

  Note: XPath 1.0 functions are valid XPath 2.0 expressions. You can use the XPath Expression Builder to generate simple XPath 1.0 expressions.
- **Database transforms**:
  - You can use the **Select** transform to query one or more database tables, and retrieve data that you can use in the message map to set output element values, define conditions, or use as input to build

other transforms conditions. Database tables can be set as additional outputs of a message map. For more information, see “Selecting data from a table” on page 1523.

– You can use a **database routine** transform to call a stored procedure or user-defined function from a database, and retrieve data that you can use in the message map to set output element values, define conditions, or use as input to build other transforms conditions.

  Note: For information about the support for stored procedures, see “Support for stored procedures” on page 1318.

What to do next

Define transformations to set the value of output elements in your output message:

- Use the **Assign** transform to set the value of an output element to a constant. For more information, see “Setting the value of an output element to a simple data type” on page 1481.
- Use any of the **xs:type** transforms to cast the value of a simple type input element and set the value of a simple type output element. For more information, see “Setting the value of an output element with a explicit data type” on page 1483.
- Use the **Create** transform to set the value of an output element that is defined as a simple type or as part of a complex data type. You use the default or the fixed value defined in the schema for that element. For more information, see “Setting the value of a simple output element to a default or fixed value” on page 1485.
- Use the **Assign** transform or the **Create** transform to create an empty output element defined as a string or as a hexBinary element. For more information, see “Creating an empty output element” on page 1489.
- Use the **Create** transform, the **Move** transform, the **Custom Java** transform, the **Custom ESQL** transform, or the return value of a database to create a null output element. For more information, see “Creating a nil output element” on page 1487.
- Use the `fn:nilled` XPath function, and the `fn:exists` XPath function to define conditional expressions that determine whether an input element is nilled, or is present. For more information, see “Choosing an XPath conditional expression that tests for a nil value in a transform ” on page 1438.
- Use the `fn:empty` XPath function to define conditional expressions that determine whether an input element is empty. For more information, see “Creating an empty output element” on page 1489.

Setting the value of an output element to a simple data type
In the Graphical Data Mapping editor, use the **Assign** transform to set the value of an output element to a constant.

About this task

You cannot use the value of an input element to set the value of an output element in an **Assign** transform.

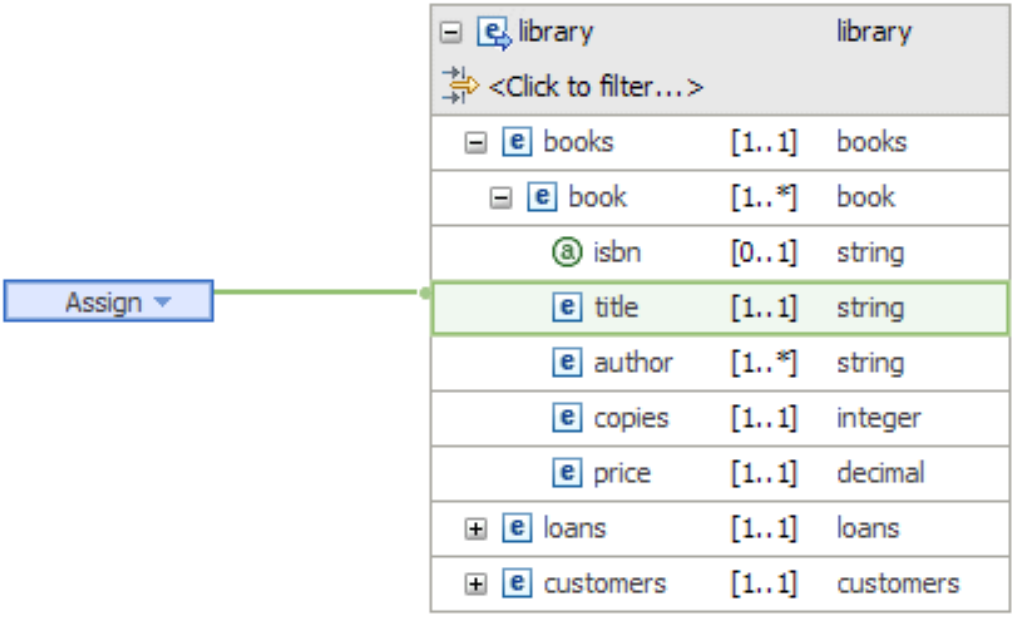
However, you can define **supplement** connections between input elements and the **Assign** transform. You can then use these input elements in a conditional expression that defines the condition under which the transform should be applied.

The output element is set to a constant value.

Procedure

Complete the following steps to set the value of an output element to a constant:

1. Create an **Assign** transform for the output element, by using either of the following methods:
   - Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
   - Right-click the output element and then select **Create Assign**.

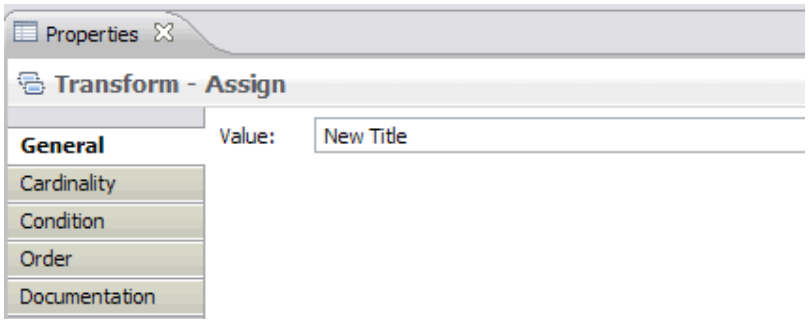


2. Enter the required constant into the **Value** field in the Properties panel for the **Assign**.

   The constant value can be any simple data type value.

   Note: Do not enclose string literals in single or double quotes; enter the required text in the entry field without quotes.

   The following figure shows the Properties view of the **Assign** transform:



Results

If you do not specify a value, an empty element is created.

By default, the following values are set for an output element when you use the **Assign** transform:

*Table 59. Default values set by using the **Assign** transform*

Type	Default value
string	Empty string
dateTime	`2002-01-01T11:00:00`

*Table 59. Default values set by using the **Assign** transform (continued)*	
Type	**Default value**
boolean	`false`
decimal	`0.0`
double	`0.0`
hexBinary	`00`
long	`0`
duration	`P1Y`
time	`00:00:00`
date	`2002-01-01`

Setting the value of an output element with a explicit data type
In the Graphical Data Mapping editor, you can use any of the **xs:type** transforms to cast the value of a simple type input element and set the value of a simple type output element.

About this task

For example, you want to assign a value with a specific data type to a target element that is defined as `xs:anySimpleType`. You can use the **xs:type** transform.

You can have zero or more input elements to an **xs:type** transform. However, you can only cast one value for an output element by using an **xs:type** transform. If the mapping for the **xs:type** transform has more than one input, any input that is not referenced by the argument of the **xs:type** transform will be ignored.

You can use the input elements to build a conditional expression that determines if the **xs:type** transform is applied or not.

You must choose the **xs:type** transform according to the output element data type. For example, if you have an output element with a boolean data type, you must choose **xs:boolean** transform.

You can use any of the following `xs:any` transforms:

- **xs:NOTATION**: This function takes a primitive and casts it as notation.
- **xs:Qname**: This function takes a primitive and casts it as Qname.
- **xs:anyURI**: This function takes a primitive and casts it as anyURI.
- **xs:base64Binary**: This function takes a primitive and casts it as base64Binary.
- **xs:boolean**: This function takes a primitive and casts it as boolean.

  You can use any of the following values: `true` or `false`.
- **xs:dateTime**: This function takes a primitive and casts it as dateTime.
- **xs:date**: This function takes a primitive and casts it as date.
- **xs:dayTimeDuration**: This function takes a primitive and casts it as dayTimeduration.
- **xs:decimal**: This function takes a primitive and casts it as decimal.
- **xs:double**: This function takes a primitive and casts it as double.
- **xs:float**: This function takes a primitive and casts it as float.
- **xs:gDay**: This function takes a primitive and casts it as gDay.
- **xs:gMonthDay**: This function takes a primitive and casts it as gMonthDay.
- **xs:gMonth**: This function takes a primitive and casts it as gMonth.
- **xs:gYearMonth**: This function takes a primitive and casts it as gYearMonth.

- **xs:gYear**: This function takes a primitive and casts it as gYear.
- **xs:hexBinary**: This function takes a primitive and casts it as hexBinary.
- **xs:integer**: This function takes a primitive and casts it as integer.
- **xs:int**: This function takes a primitive and casts it as signed 32-bit integer.
- **xs:string**: This function takes a primitive and casts it as string.
- **xs:time**: This function takes a primitive and casts it as time.
- **xs:yearMonthDuration**: This function takes a primitive and casts it as yearMonthDuration.

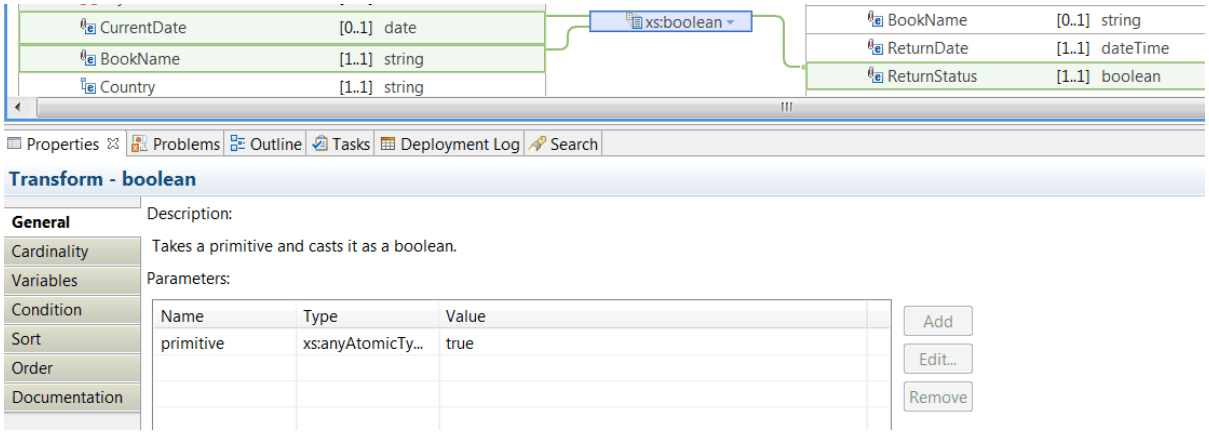
Procedure

Complete the following steps to set the value of a output element by using an **xs:type** transform:

1. Add an **xs:type** transform, for example, an **xs:boolean** transform, to set the value of the output element, by using either of the following methods:
   a) Create an **Assign** transform by using either of the following methods:
      - Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
      - Right-click the output element and then select **Create Assign**.

   b) Use the drop-down, and select **Cast Function** > **xs:boolean**.
2. Enter `true` into the **Value** field in the Properties panel for the **xs:boolean** transform.

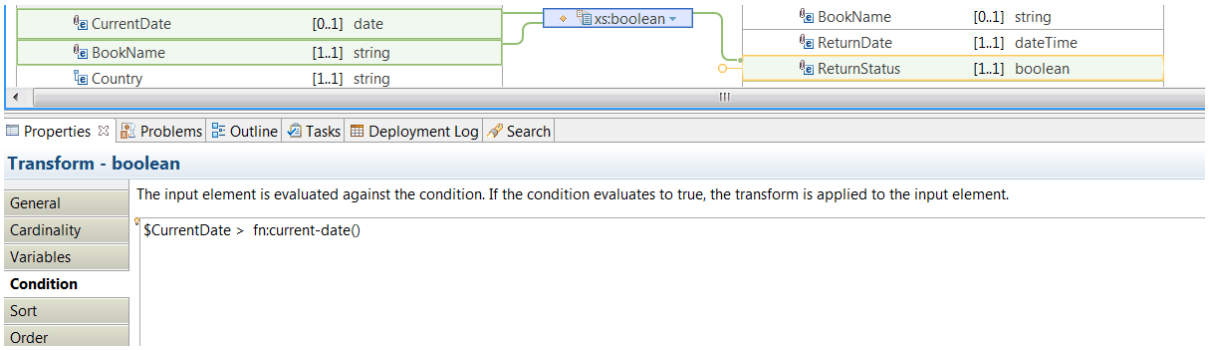
   You can set the value to `true` or `false`.



3. Optional: Define a conditional expression to control when the transform should be applied .

   You define a conditional expression by using an XPath expression.

   For example, to evaluate the transform, the current date has to be a date after the date specified in the element **CurrentDate**.



Setting the value of a simple output element to a default or fixed value
In the Graphical Data Mapping editor, use the **Create** transform to set the value of a simple output element. You use the default or the fixed value defined in the schema for that element.

About this task

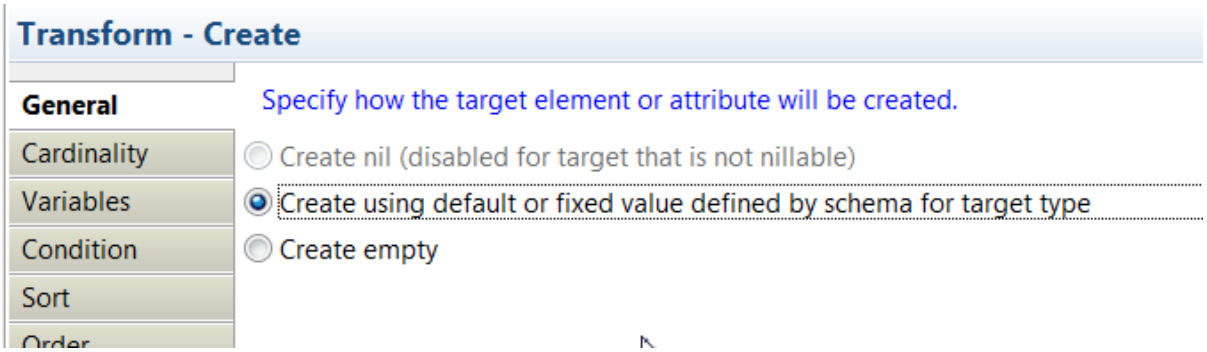
The **Create** transform creates an output element without the use of input data.

Note: You can only set the value of a simple output element to its default or fixed value when the schema includes a default value or a fixed value for the element.

Procedure

Complete the following steps to create a simple output element with its default or fixed value:

- Create an **Assign** transform by using either of the following methods:
  a) Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
  b) Right-click the output element and then select **Create Assign**.
- Use the drop-down, and select **Core transforms** > **Create**.
- In the Properties panel, select the **General** tab. Then, select **Create using default or fixed value defined by schema for target type**.



Setting the value of a simple type element included in a complex type output structure to a default or fixed value
In the Graphical Data Mapping editor, use the **Create** transform to set the value of an output element that is defined as part of a complex data type. You use the default or the fixed value defined in the schema for that element.

About this task

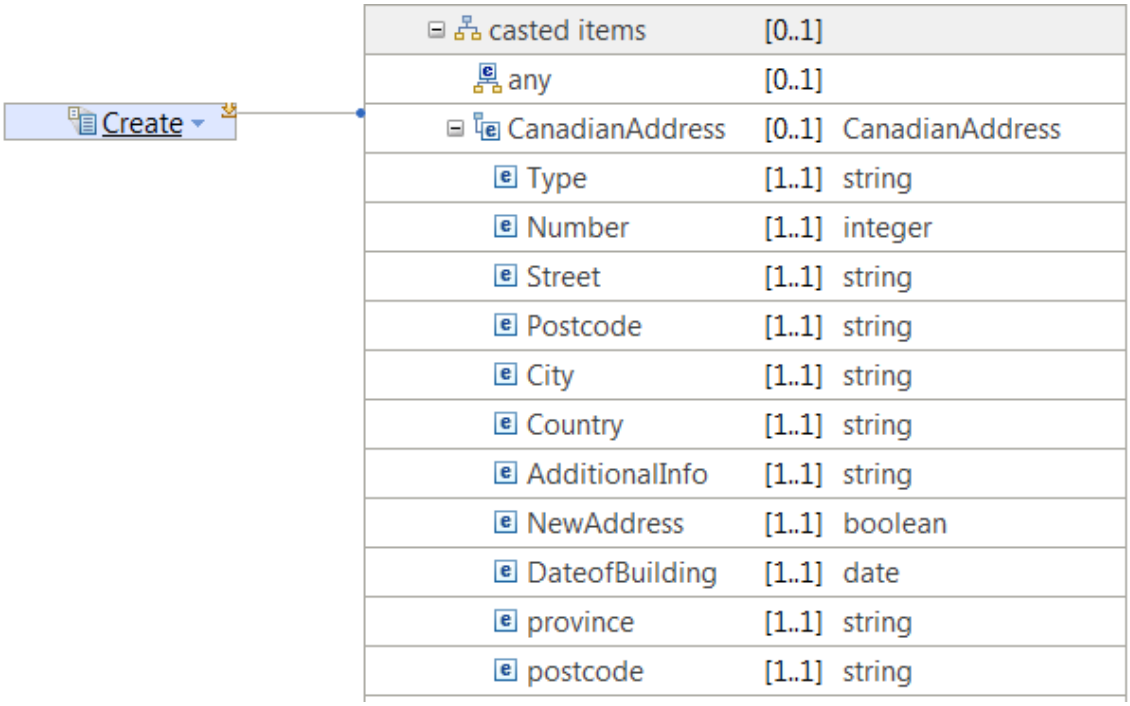
The **Create** transform creates an output element without the use of input data.

Note: You can only set the value of a simple output element to its default or fixed value when the schema includes a default value or a fixed value for the element.

Procedure

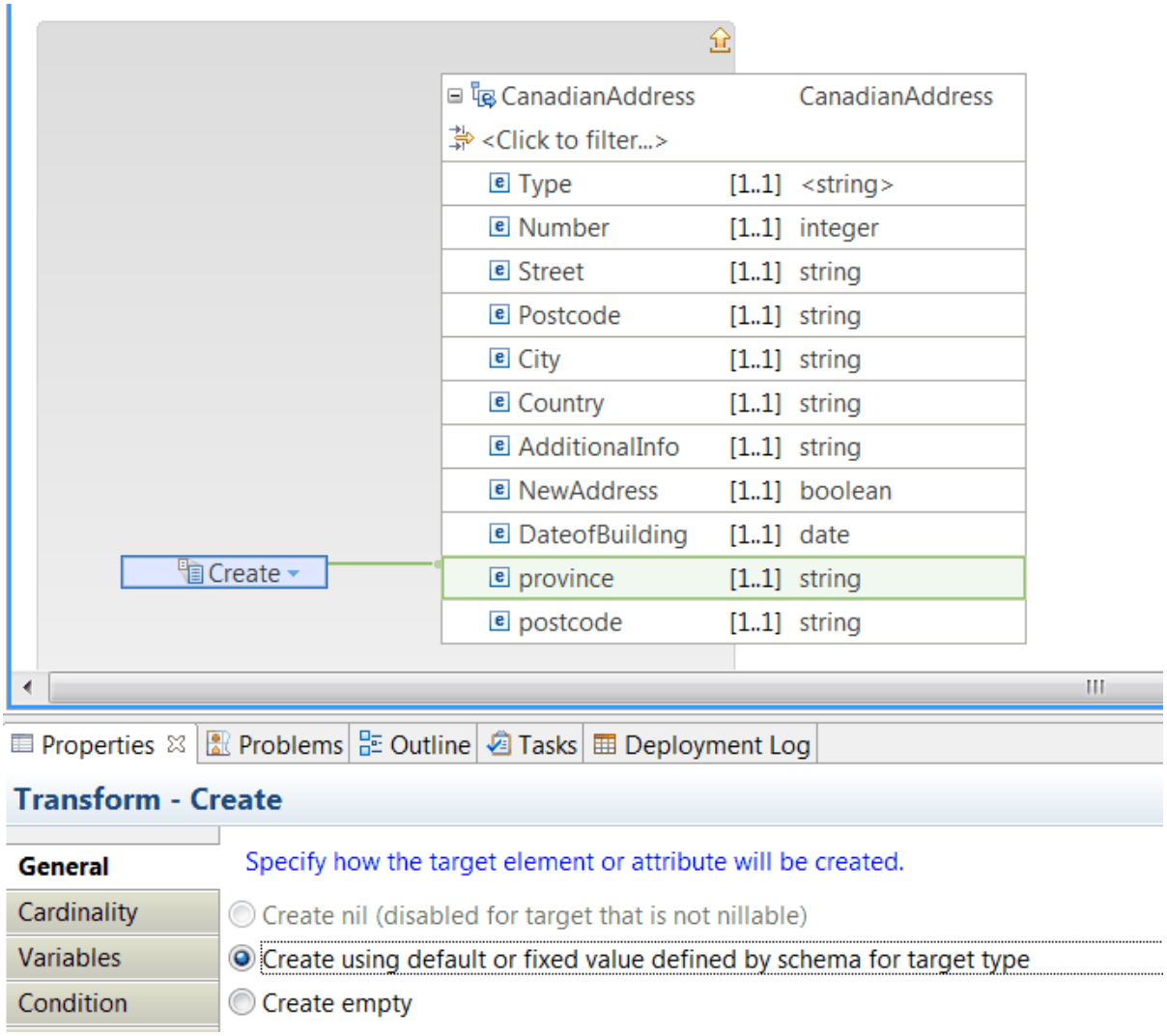
Complete the following steps to create a simple type element included in a complex type output structure with its default or fixed value:

- Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, a **Create** transform is created.



- Open the nested map associated to the **Create** transform. You can define how each target element or attribute will be created within the complex element. For more information, see "Setting the value of a simple output element to a default or fixed value" on page 1485.

  You can use in the **Create** transform nested map more **Create** transforms, **Assign** transforms, or any other mapping transforms that do not require an input. You can use this method to populate as many fields as required in the complex output structure.



Creating a nil output element
In the Graphical Data Mapping editor, you can use the **Create** transform, the **Move** transform, the **Custom Java** transform, the **Custom ESQL** transform, or the return value of a database to create a nil output element.

Before you begin

When you map null values, consider the Graphical Data Mapping editor behavior. For more information, see "Handling nulls in message maps" on page 1368.

Procedure

Choose any of the following methods to create a nil output element:

- Use the **Create** transform. Create a nil element without the use of input data.
- Use the **Move** transform. Copy an input element that is nil to an output nil element.

- Use the **Custom Java** transform.

  The Java method that you implement must return an MbElement that is nil.
- Use the **Custom ESQL** transform. Your ESQL code must return a NULL value.
- Use the database transform, and **Move** a nullable column to an output nillable element.
- Use a **Custom XPath** transform with the function `iib:nullValue()` to set an XMLNSC element to `nil`, a JSON object to NULL, and a JSON array to NULL.

Example

Example: Use the Create transform to create a simple type output element as nil

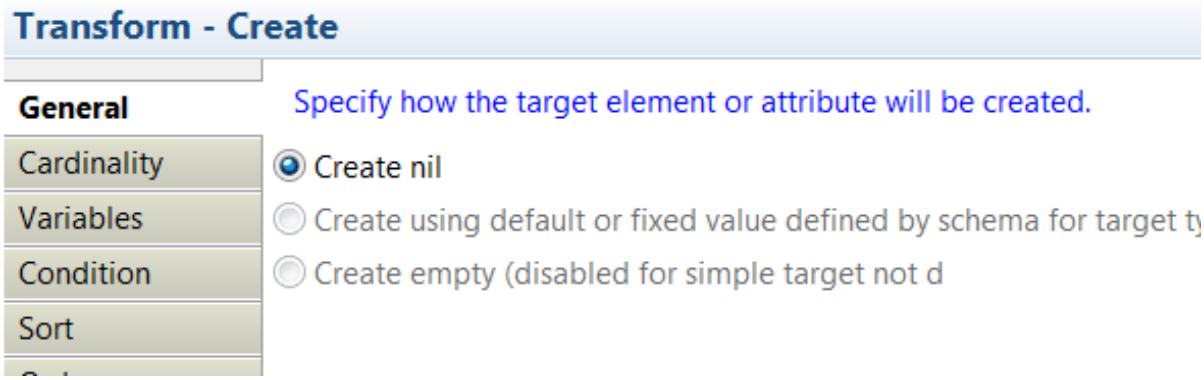
This example shows how to use the **Create** transform to create a simple type output element as nil in the Graphical Data Mapping editor.

You can use the **Create** transform to create an output element with `xsi:nil="true"`, also called a nil element, without the use of input data.

Note: The option to create a nil element is available only when the output element is nillable.

Complete the following steps to create a simple nil output element:

1. Add a **Create** transform to set the value of the output element.
   a. Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
   b. Select the drop-down, and select **Core transforms** > **Create**.
2. In the Properties panel, select the **General** tab. Then, select **Create nil**.



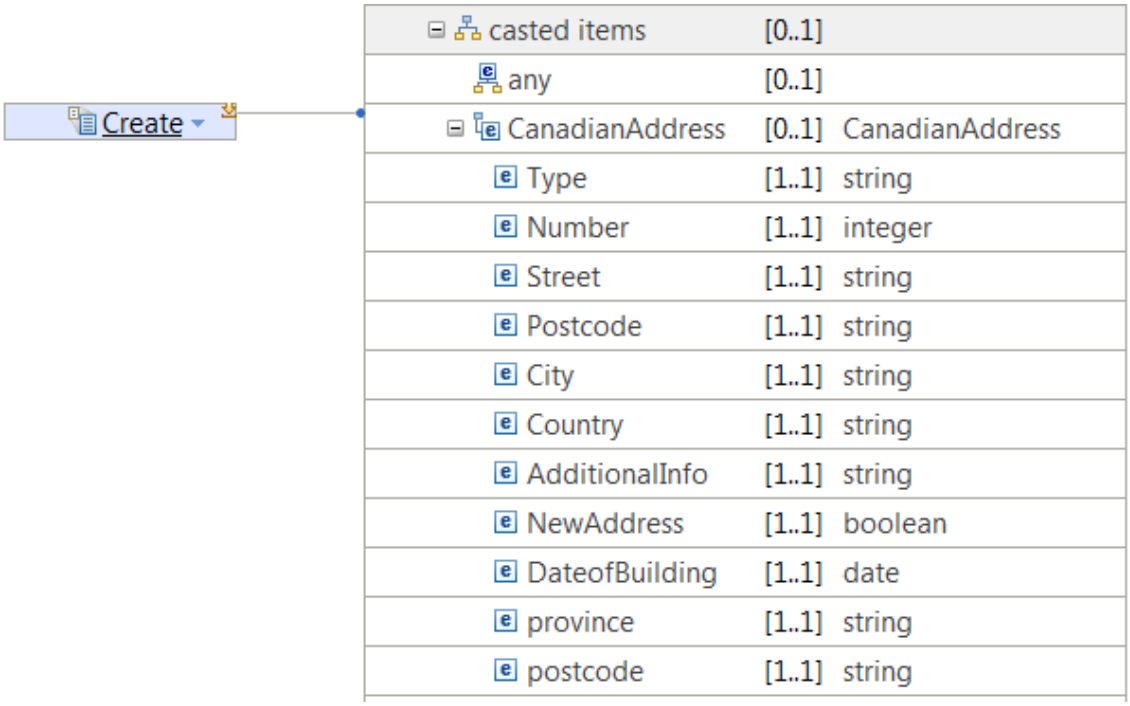
Example: Use the Create transform to create a complex type output element as nil

This example shows how to use the **Create** transform to create a complex type output element as nil in the Graphical Data Mapping editor.

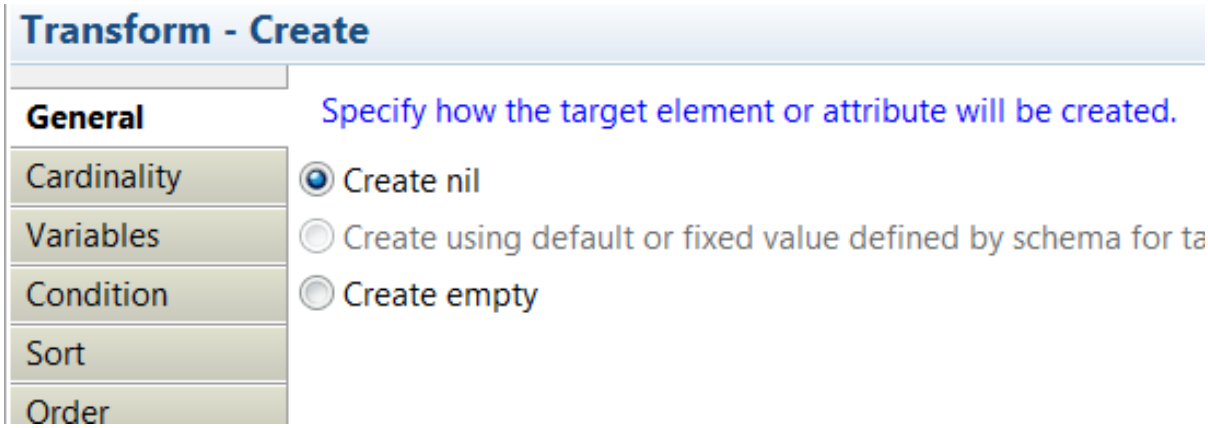
Note: The option to create a nil element is available only when the output element is nillable.

Note: The **Create** transform has a nested map, however, you cannot create or assign fields within a complex nil output element. You will get the following error: `Cannot create or assign fields within nil target. To allow mapping, change parent Create transform to create empty instead of nil.`

1. Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, a **Create** transform is created.



2. In the Properties panel, select the **General** tab. Then, select **Create nil**.



Creating an empty output element
In the Graphical Data Mapping editor, you can use the **Assign** transform or the **Create** transform to create an empty output element defined as a string or as a hexBinary element when you have no source to copy from.

About this task

You can define an empty output element when the following conditions are met:

- The data type of the element is string or hexBinary.
- The element is not nillable.

Procedure

Choose any of the following methods to create an empty output element:

1. Use the **Assign** transform to create an empty output element. For more information, see “Initializing an output element by using the Assign transform” on page 1490.
2. Use the **Create** transform to create a simple or complex empty output element. For more information, see “Initializing a simple or complex output element by using the Create transform” on page 1491.

Results

The following table lists the data types and the transforms that you can use to create an output element:

Table 60. List of transform types that you can use to create an empty element

Data type	**Valid transforms**
Simple data types: string, hexBinary	Assign, Create
Complex data types	Create

*Initializing an output element by using the **Assign** transform*
In the Graphical Data Mapping editor, use the **Assign** transform to create an empty output element.

About this task

There is no input element to an **Assign** transform.

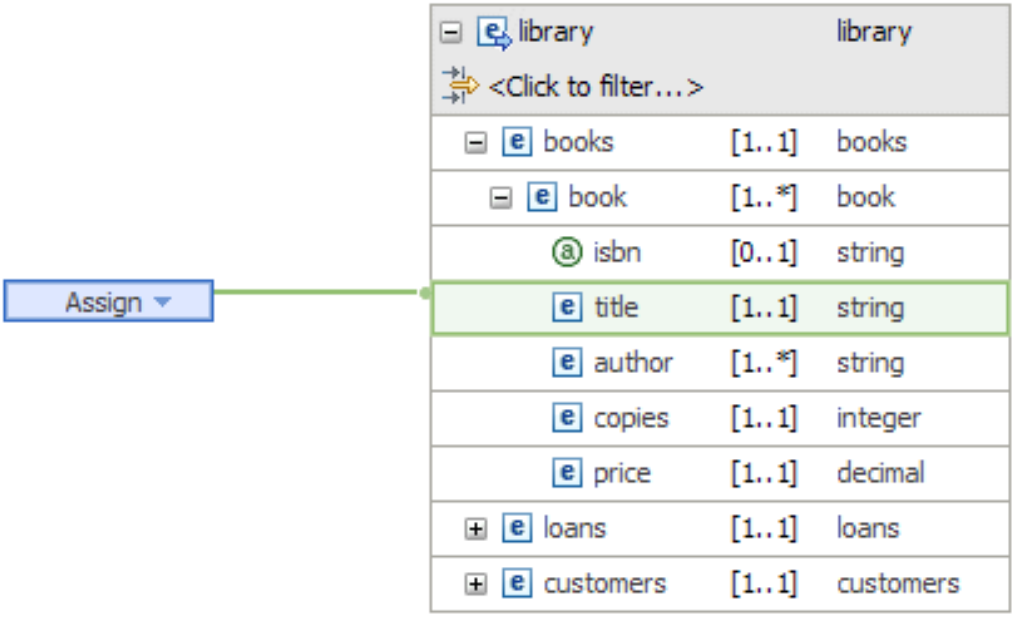
You can only create an empty output element for the following output element data types:

- string
- hexBinary

Procedure

Complete the following steps create an empty output element:

1. Create an **Assign** transform for the output element, by using either of the following methods:
   - Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
   - Right-click the output element and then select **Create Assign**.



2. Do not specify a value in the **Value** field in the Properties panel for the **Assign** transform.

*Initializing a simple or complex output element by using the **Create** transform*
In the Graphical Data Mapping editor, use the **Create** transform to create a simple or complex empty output element.

About this task

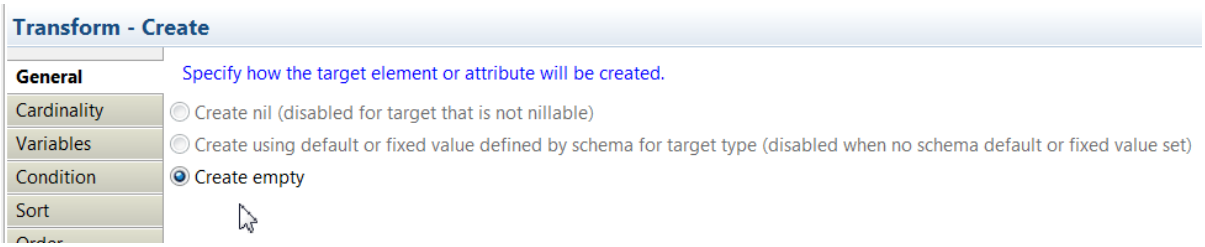
The **Create** transform creates an output element without the use of input data. You can create either simple or complex type output elements.

You can create a simple empty output element for elements defined as string or hexBinary.

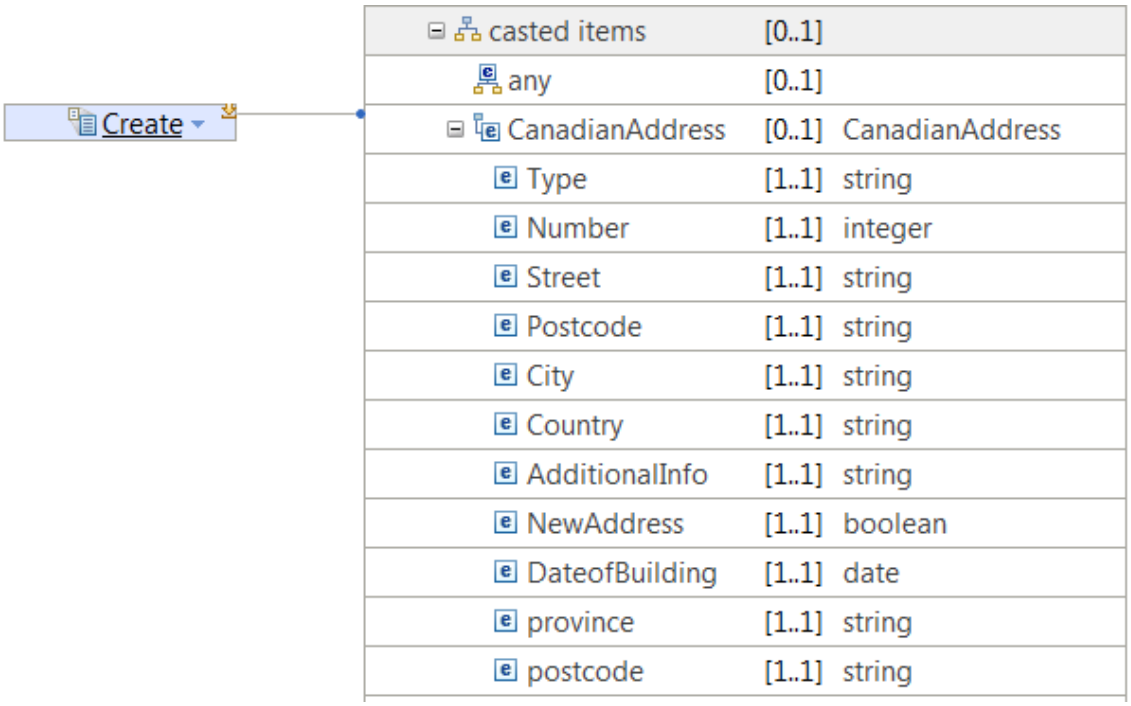
Procedure

Complete the following steps to create a simple or complex empty output element:

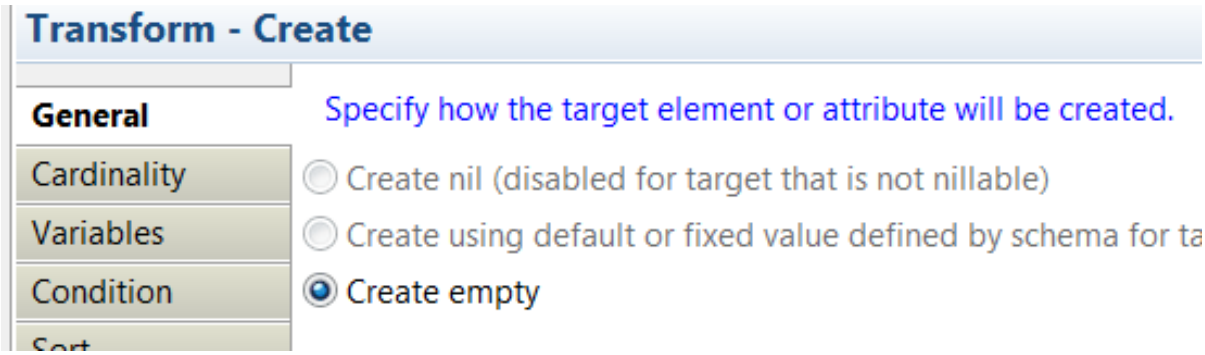
- For simple type output elements, complete the following steps:
  a) Add a **Create** transform to set the value of the output element, by using either of the following methods:
     – Create an **Assign** transform by using either of the following methods:
       - Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, an **Assign** transform is created.
       - Right-click the output element and then select **Create Assign**.
     – Use the drop-down, and select **Core transforms** > **Create**.
  b) In the Properties panel, select the **General** tab. Then, select **Create empty**.



- For complex type output elements, complete the following steps:
  a) Click the output element and drag the connector towards the input elements, releasing the connector onto a blank part of the canvas. When you release the connector, a **Create** transform is created.



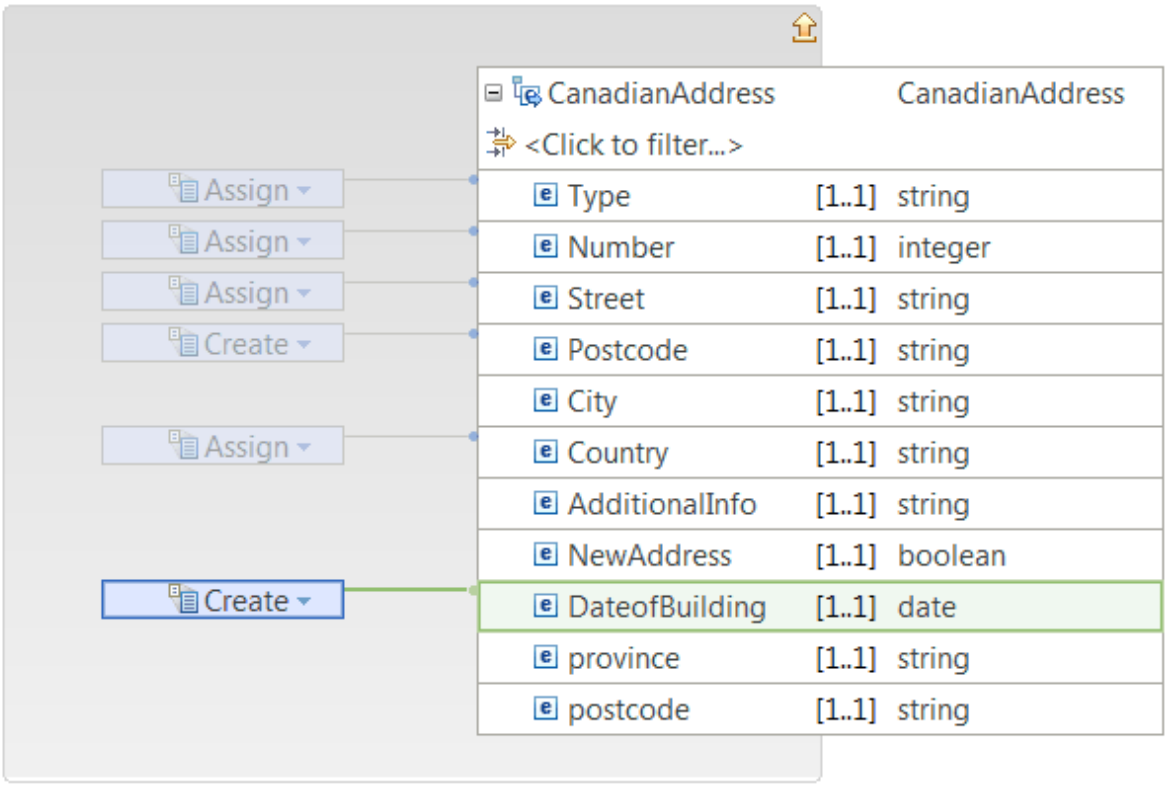
  b) In the Properties panel, select the **General** tab. Then, select **Create empty**.

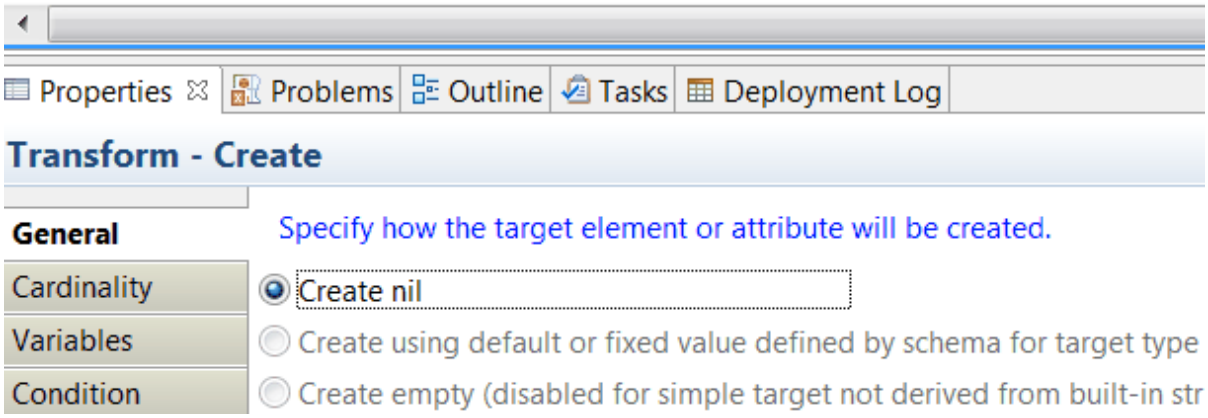


  c) Optional: Open the nested map associated to the **Create** transform. You can define how each target element or attribute will be created in a complex element.

  You can use the **Create** transform to create an empty complex output element. then, you can enter the **Create** transform nested map and use more **Create** transforms, **Assign** transforms, or any other

mapping transforms that do not require an input. You can use this method to populate as many fields as required in the complex output structure.





Checking whether the value of an input element exists, or is null

You can use the `fn:exists` XPath function, and the `fn:nilled` XPath function to define conditional expressions that determine whether an input element is present, or is null.

About this task

You can define conditional expressions in a transform to determine whether the transform should be applied or not in your message map.

For example, you might decide not to apply a transform if the input value is null.

You can use the following XPath functions to define conditional expressions in a transform:

- Use the `fn:nilled` XPath function to test whether the value of an input element has the `xsi:nil` attribute set.

- Use the `fn:exists` XPath function to test whether the value of an input element is present.

  Note: An XML element that has the `xsi:nil` attribute set is considered to be present.

Procedure

Take the following steps to define a conditional expression in a **Move** transform that determines whether the transform is applied or not. The transform applies if the element exists, and is not null.

1. Create a **Move** transform between an input element and an output element.
2. In the Properties view, under **Condition**, enter an XPath expression to define the conditional expression that must evaluate to true for the transform to be applied.

```
fn:exists($BookName) and (fn:nilled( $BookName) = false)
```

Checking whether the value of an input element is empty

Use appropriate expressions to test for empty input values, depending on whether the element is a simple or complex type.

About this task

You can define conditional expressions in a transform to determine whether the transform is applied or not in your message map. For example, you might decide not to apply a transform if the input value is empty.

For simple string types, you can test whether the element is empty by comparing with the empty string `''` (two single quotation marks). For example, to check whether a string element that is represented by the variable name `str` is empty, use the following expression:

```
$str = ''
```

If the element is also nillable, use the following expression:

```
(fn:nilled( $elementName) = false)  and $str = ''
```

Other simple types have no natural empty value. For example, with numerical type elements the value must always contain a minimum set of valid digits. In this case, you might consider a particular value to be empty, and so test for that value. To check whether a number element that is represented by the variable name `num` has a zero value, use the following expression:

```
$num = 0
```

To determine whether a complex element is empty, check for child elements or attributes. For example, to check a complex element that is represented by the variable name `comp`, use the following expression:

```
fn:not( $comp/* or $comp/@* )
```

If the complex element allows mixed content, check for text content as well.

```
fn:not( $comp/text() or  $nodes/* or $nodes/@* )
```

Procedure

Take the following steps to define a conditional expression on a **Move** transform between a source and target string element. The expression ensures that the transform is not applied if the input data is an empty string.

1. Create a **Move** transform between an input element and an output element.

2. In the Properties view, under **Condition**, use content assist (Ctrl+Space) to enter the appropriate XPath expression.
   For example, assuming the variable name for the input is `str`:

```
$str = ''
```

Mapping an element by using a Custom Java transform

In the Graphical Data Mapping editor, you can use a method in a Java class to set the value of an output element. You can use the Custom Java transform to invoke a Java method, optionally passing one or more values to its parameters.

Procedure

Complete the following steps to use Java in a Custom Java transform:

1. Create a message map.

   When you use Java code in a map, you can define the map in a static library project, a shared library project, an application project, or an integration project. You can define the Java code in one or more Java projects. Your map project must reference these Java projects. For more information, see Adding a project to an application, integration service, or library.

   Note: When the map uses Java code available in a shared library, you must create the map in the shared library project where the Java code is available.

   For more information, see “Creating message maps” on page 1378.
2. Add a Custom Java transform to the map, and connect an input and output:
   - The input and output elements must be child elements; you cannot use a Custom Java transform to map a root element of a message.
   - Ensure that the input elements provide simple data values, and the output element has a data value set by the invoked Java method.
   - If you want to map complex or repeating elements in a Custom Java transform, see “Processing complex or repeating elements in a Custom Java transform” on page 1498.

3. If you have already defined a Java class, select the class in the **General** tab of the **Properties** page of the transform or, if you have not defined a Java class, complete the following steps to define the required Java class:
   a) In the **General** tab of the **Properties** page of the transform, click **New**.
   b) Select one of the following options, and complete the associated steps:
      - Create a new Java project to store the class:
        i) Select **Create the class in a new Java project to be created** and click **Next**.
        ii) Complete the fields in the dialog (for more information, see Creating a Java project) and click **Next**.
      - Use an existing Java project to store the class:
        i) Select **Create the class in one of the existing Java projects**.
        ii) Select the Java project from the list and click **Next**.

      The **New Java class wizard** is displayed.
   c) Complete the **New Java class wizard**.
      At a minimum, complete the following fields:
      i) In the **Source** field, select the `src` folder of your Java project.
      ii) In the **Package** field, enter the name of a package to contain the new class.
      iii) In the **Name** field, enter a name for the class.
   d) Click **Finish**. If prompted, click **Yes** to create a reference from the application that contains your map to your Java project.
      The Java class is created with a sample method.
   e) In the **General** tab of the **Properties** page of the transform, click **Edit** to complete your Java class.

      The Java class that you provide to the map must have static methods that return the appropriate type for the value of the output element, and take parameters of the appropriate type for the wired inputs.

      You can use the MbElement class in a Java method to pass a complex type, a wildcard **xsd:any**, or a wildcard **xsd:anySimpleType**. For more information, see "Custom Java" on page 1312.

      For example, the following Java method might be used in a Custom Java transform that has three input elements, of types a `xs:string`, `xs:decimal`, and `xs:boolean` and the output element is a `xs:decimal`:

```
public static BigDecimal calSomething(String memType, BigDecimal stdCost, boolean flag) {
 BigDecimal actualCost = stdCost;
 if (flag & memType.startsWith("gold")) {
  BigDecimal discRate = new BigDecimal(0.9);
  actualCost = actualCost.multiply(discRate);
 }
 return actualCost;
}
```

      Note: If the input element that is used to provide a value for a Java method is not of the correct type, you can use a type cast function, for example **xs:int($var)**, to set the required type. For more information, see "Cast type (xs:type)" on page 1299.

4. Configure the method that defines the transformation logic that is applied by the Custom Java transform. In the **General** tab of the **Properties** page of the transform, complete the following steps:
   a) Click **Browse** and complete the following steps:
      i) In the **Select Type** window, enter the name of the Java package in the **Select entries** field. The package must contain the Java class.

         Note: You must start typing the name of your Java package before you can see any classes to choose from in the **Select entries** field.

      ii) Select a Java class and click **OK**.

   Note: If the Java project that contains the Java class does not build in Eclipse, then the Java class is not visible anywhere in the map. You must resolve all the Java errors before you can configure the class and method in the **General** tab of the **Properties** page. You can see the errors in the **Problems** tab.

   b) Configure one Java method. You access the methods that are available through the drop-down.

      If the method has parameters, they are added automatically in the **Parameters** section of the **General** tab.

   c) Set the value of each parameter. Complete the following steps to set the value of a parameter:
      i) Select a parameter.
      ii) Click the **Value** column of a parameter.
      iii) Select an element, or select the option **Edit custom XPath expression parameter**. If you select the edit option, you can define an XPath expression or a call to a static method on an imported Java class that is visible to the map. You can also create a complex expression that includes XPath, Java, and extension functions such as **iib:getUserDefinedProperty("propertyname")**.

   When you define a Java method and a Java class in the **General** tab of the **Properties** page of a **Custom Java** transform, an import is automatically added in the map to refer to the package qualified Java class. A prefix based on the class name is added. All the **public static methods** in this Java class are then available in content-assist when building expressions in other transforms.

   Alternatively, you can use Java class methods that use standard Java types in any expression within your map, for example as part of the condition of an `if` transform. Before you can use Java methods in expressions, import the Java class into the map. The method is then available in content assist (`Ctrl-space`). To import a Java class into a map, configure the **Java imports** tab in the **Properties** page of the map.

What to do next

Deploy and test the message map. For more information, see “Troubleshooting a message map” on page 1587.

Processing complex or repeating elements in a Custom Java transform
You can use the Java MbElement class for mapping inputs and outputs that are not simple types with a **Custom Java** transform.

Using the MbElement class

To use the MbElement class in the Java code that you use in a map, complete the following tasks:

1. Add the MbElement plug-in (**jplugin2.jar**) to the build path of the Java project.

   The plug-in is a JAR file that is provided with the IBM App Connect Enterprise Toolkit. The **jplugin2.jar** is also available in the classes directory of the server installation.

   a. In the Java perspective, right-click the Java project, and select **Properties.**
   b. Select **Java Build Path**.
   c. Select the **Libraries** tab.
   d. Click **Add Variable**.
   e. Select the variable **JCN_HOME** and click **Extend**.
   f. Select **jplugin2.jar**, and click **OK**.
2. Import the MbElement class into your Java source.

   You must add the following code:

```
import com.ibm.broker.plugin.MbElement;
```

Using the Java DOM API

You can use the **Node** class from the standard Java **DOM API** to process complex or repeating elements in a Custom Java transform.

For example, the following code shows a Java method that uses the **Node** class:

```
public static Node nodeMove(Node inEl)
```

Reusing Java code

If your Java code is likely to be used by multiple solutions, store it in a shared library. You can store the Java code in the same shared library as a message map. Alternatively, you can store the Java code separately in a referenced shared library.

When you define transformations in your map that use Java code, you can define a map in a static library project, a shared library project, an application project, or an integration project. You can define the Java code in one or more Java projects. Your map project must reference these Java projects.

Note: When the map uses Java code available in a shared library, you must create the map in the shared library project where the Java code is available.

Mapping a single non-repeating element

When you map a single non-repeating element input to a single non-repeating element output, you can use a Java method with the following signature:

```
public static MbElement mbElMove(MbElement inEl)
```

For example, the following code shows a Java method that copies a sub tree:

```
public static MbElement mbElMove(MbElement inEl)
      {
  MbElement outEl = null;
  try {
   outEl = inEl.copy();
   outEl.copyElementTree(inEl);
  } catch (MbException e) {
   throw (new RuntimeException(e));
  }
  return outEl;
 }
```

Mapping a single repeating element

When you map a single repeating element input to an output repeating element, you can use a Java method with the following signature:

```
public static List<MbElement>; customCompleTypeMove(List<MbElement>; inEls)
```

For example:

```
public static List<MbElement> customCompleTypeMove(List<MbElement> inEls)
      {
  List<MbElement> outEls = new ArrayList<MbElement>();
  try {
   Iterator<MbElement> i = inEls.iterator();
   while (i.hasNext()) {
    MbElement inEl = i.next();
    MbElement outEl = inEl.copy();
    // Do some processing of outEl
    outEls.add(outEl);
   }
  } catch (MbException e) {
   throw (new RuntimeException(e));
  }
  return outEls;
 }
```

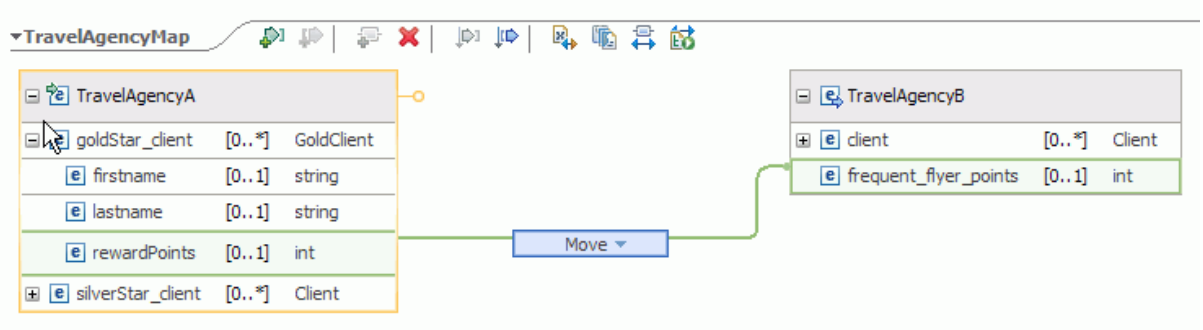
Copying a selected element of a repeating structure to a single output

In the Graphical Data Mapping editor, to copy one element from the input repeating element (array) to an output simple element, use the **Move** transform.

Procedure

1. Define a **Move** transform between the input and the output elements.

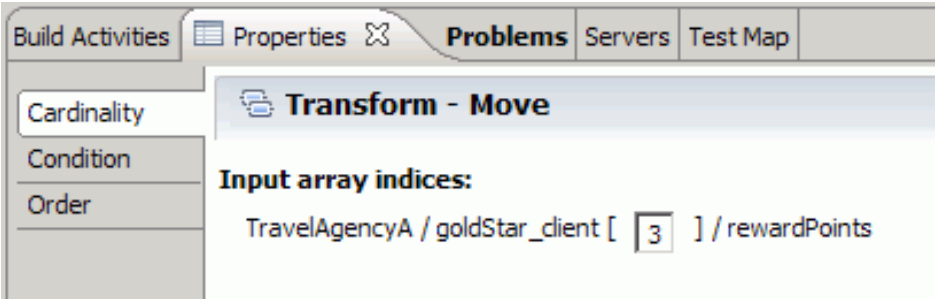
   The following figure shows the input and output elements connected by a **Move** transform:



2. Specify the index of the repeating structure that you must copy to the output.
   a) Click the **Move** transform
   b) Switch to the Properties view.
   c) In the **Cardinality** properties page, enter a value in the **Input array indices** field.

      You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see "Selecting the indexes of input array elements" on page 1363.

      For example, enter **3** to copy the third element in the repeating structure (array).



      There is no option to set the cardinality of the output element, because it is a single element.

Copying some values of a repeating element when the input and output structures are the same

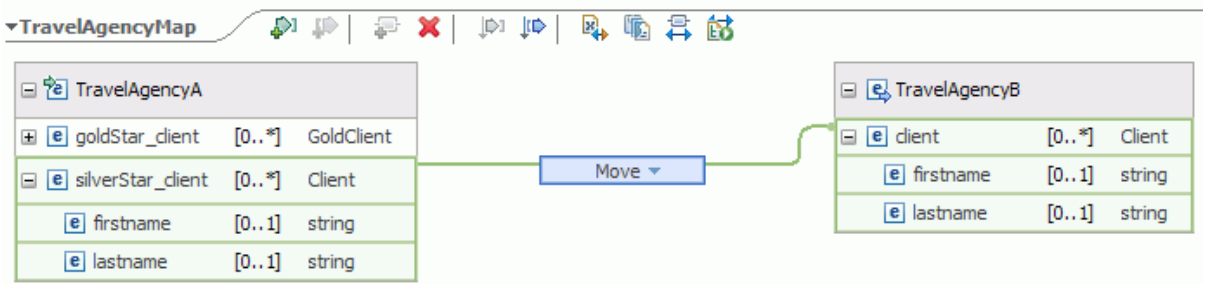
In the Graphical Data Mapping editor, to move selected elements of the input repeating element (array) to an output repeating element of the same type, select a **Move** transform between the elements, and then select the required elements in the **Cardinality** properties page.

Procedure

Complete the following steps to copy some elements of an input array to an output array when both repeating structures are identical:

1. Define a **Move** transform between the input and the output repeating structures.

   The following figure shows the input and output repeating structures of type **Client** connected by a **Move** transform:

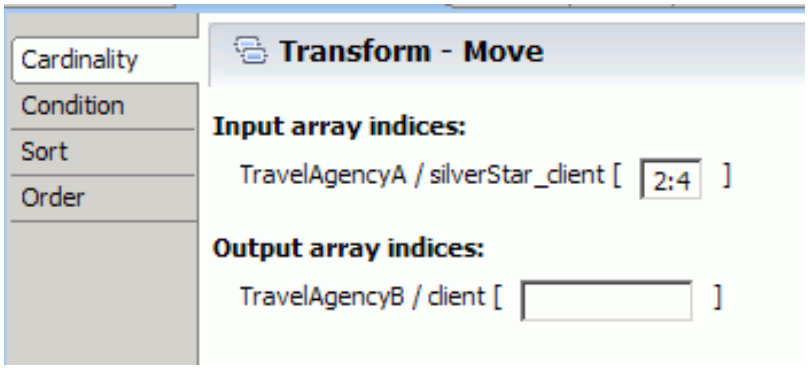


2. Specify the indexes that should be copied to the output repeating structure.
   a) Click the **Move** transform
   b) Switch to the Properties view.
   c) In the **Cardinality** properties page, enter a value in the **Input array indices** field.

      You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see “Selecting the indexes of input array elements” on page 1363.

      For example, to select elements 2, 3, and 4 of the array, set the cardinality to **2:4**.

      The first index element has a cardinality of 1.



Copying some values of a repeating element when the input and output structures are different

In the Graphical Data Mapping editor, use the **For Each** transform to iterate over an input repeating element. Then, to determine the value of the output elements, define transforms in the nested map associated with a **For Each** transform.

About this task

Use the **For Each** transform to move each element of an input array into an output array. The input and output arrays can be of the same type or of different types. For more information, see “For Each” on page 1320.

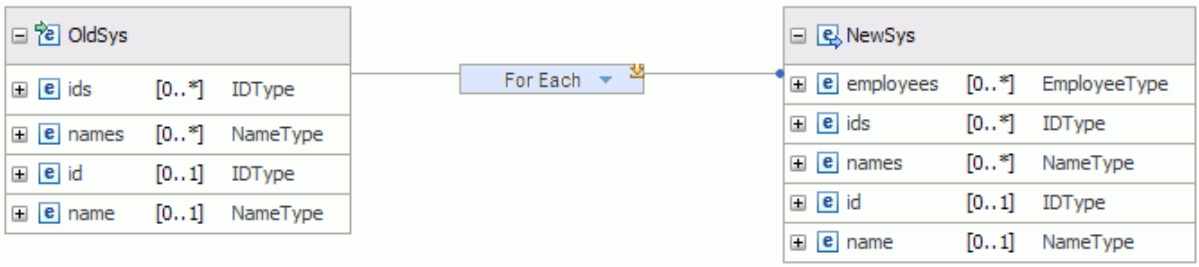
Iteration is performed over the input array.

The nested map associated to a **For Each** transform defines the type of transformations that will be performed for each iteration.

Procedure

Complete the following steps to copy some elements of an input array to an output array when the input and the output repeating elements have different types:

1. Define a **For Each** transform between the input and the output repeating structures.



2. Optional: Specify the indexes of the input repeating structure over which to iterate.
   a) Click the **For Each** transform
   b) Switch to the Properties view.
   c) In the **Cardinality** properties page, enter a value in the **Input array indices** field.

      You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see "Selecting the indexes of input array elements" on page 1363.
3. Edit the nested map by clicking the **For each** transform.
4. Define the transformations required to set the value of the output elements.

   For more information, see "Transforms (Mapping operations)" on page 1280.

Splitting an input message into multiple output messages

You can use a **For Each** transform or a **Join** transform wired to the head of the output message assembly to create a map that takes a single input message and produces multiple instances of an output message model. A typical use of this function is message splitting, in which an input batch message is divided into individual record messages.

About this task

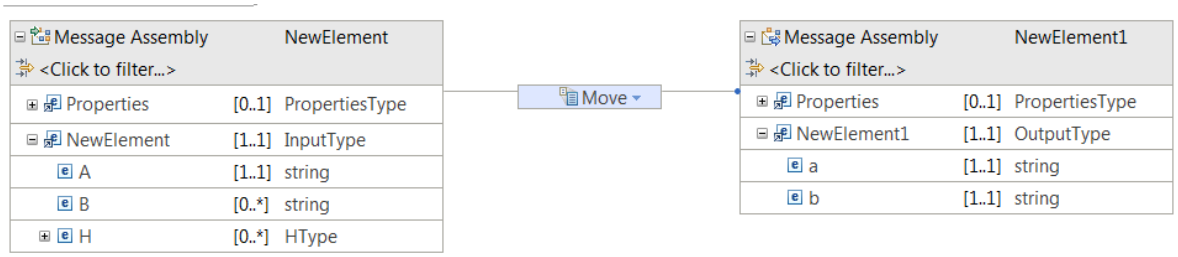
When the map runs, a new message is propagated for each iteration of the **For Each** or **Join** transform.

Procedure

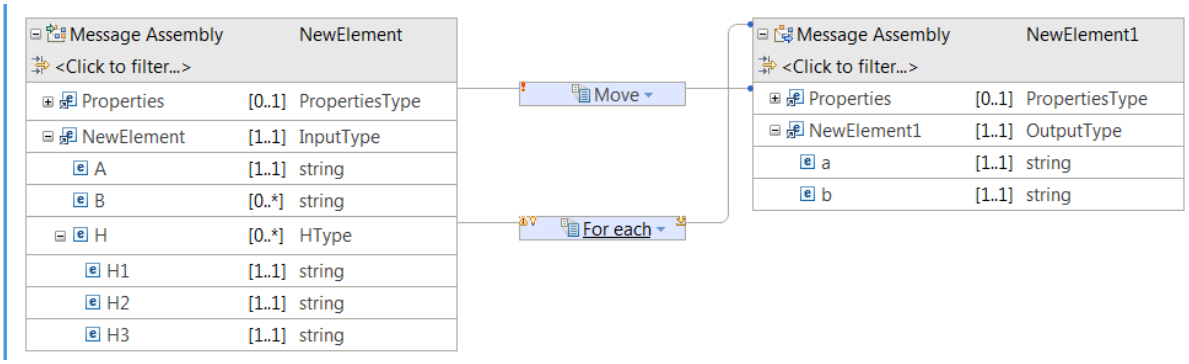
Complete the following steps to split a message into multiple output messages by using the **For Each** transform:

1. Create a map, and add one input element to your input message assembly, and one output element to your output message assembly.

   The input message assembly must contain a repeating element. The values of the elements for each index are used to populate each output message instance.

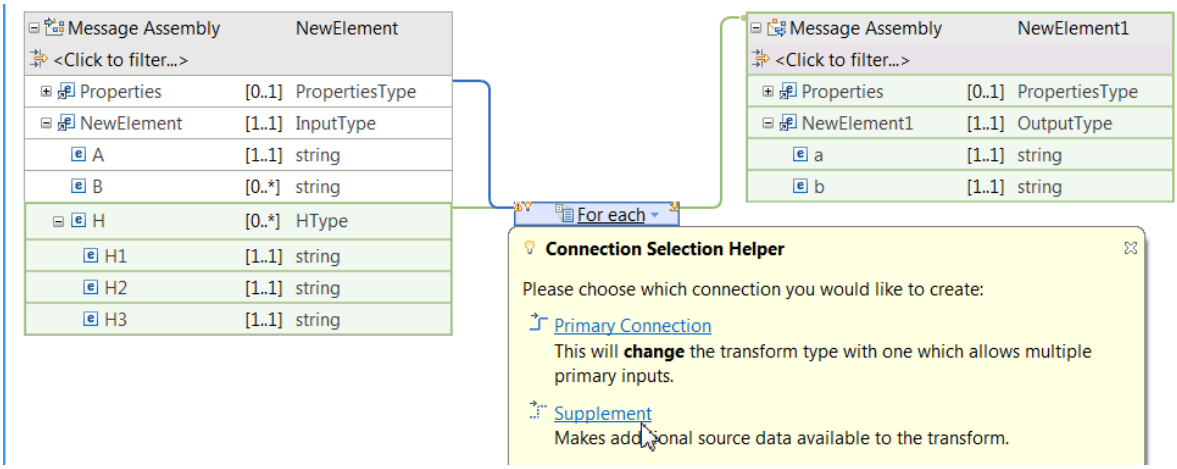


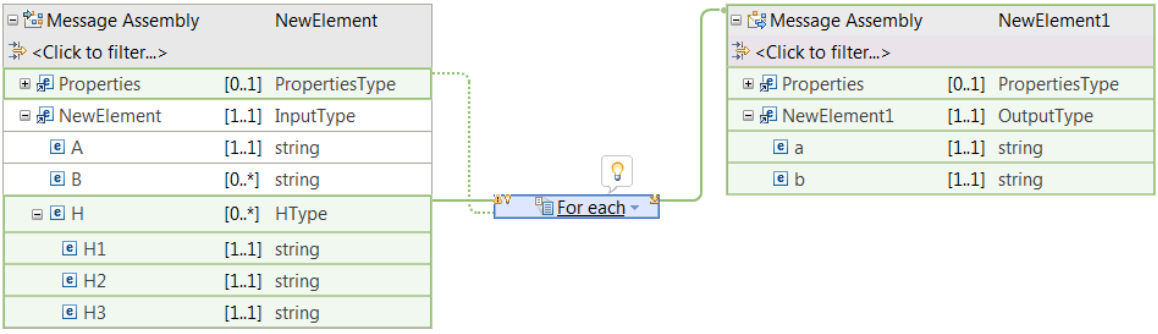
2. Define a **For Each** transform between the input element and the head of the output message assembly.



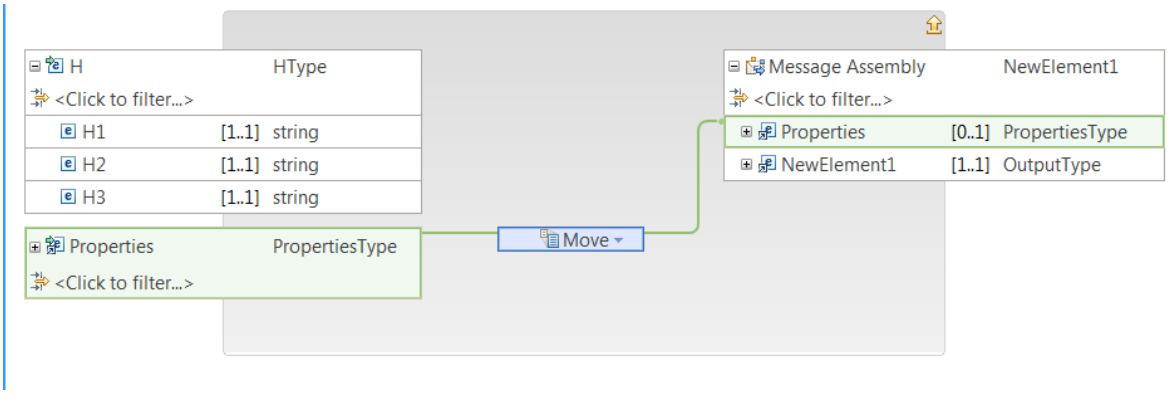
The **Move** transform between the input and the output Properties tree will display an error. Continue with the steps to remove the error.

3. Delete the **Move** transform, and then connect with a supplement connection the input Properties tree to the **For Each** transform.





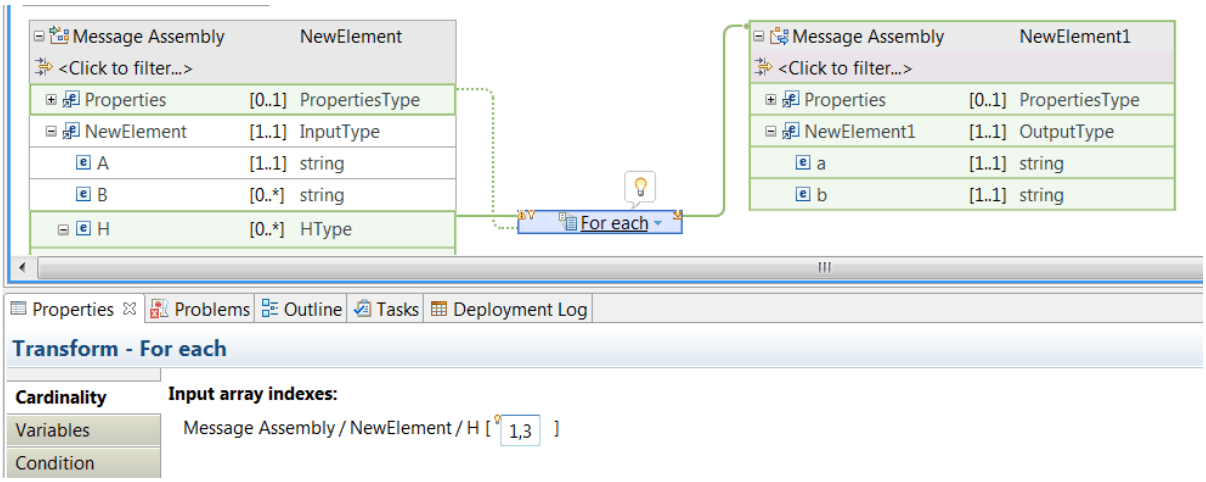
4. Select the **For Each** transform to open the nested map. Then, define a **Move** transform between the input and the output Properties tree.



5. Optional: Return to the **For Each** transform, and configure the indexes of the **For Each** transform for which you want to generate an output message.

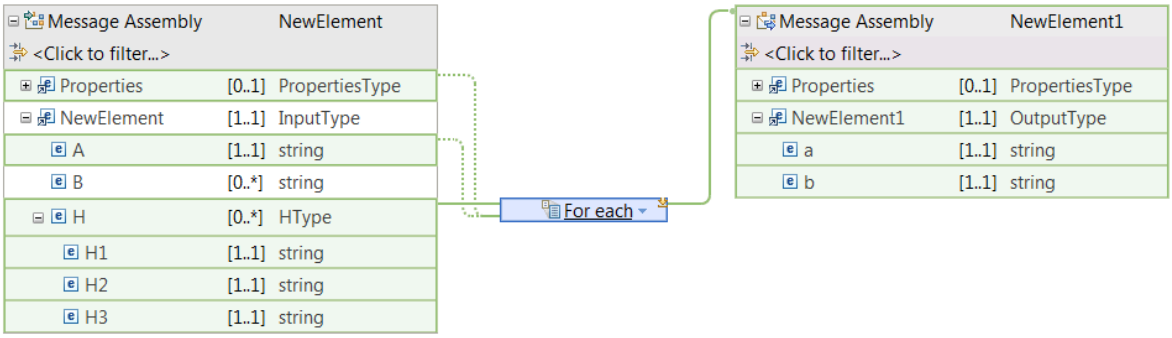
   If you need to create an output message for each index of the repeating structure, then continue with the next step.

   You can configure the **Input array indexes** section to select specific instances of the input array. For more information, see "Selecting the indexes of input array elements" on page 1363.



6. Optional: If you need additional information from the input message to populate output elements, define a supplement connection between each input element to the **For Each** transform.

   When you connect additional input elements with supplement connections to a **For Each** transform, you can use these elements as part of your mapping operations within the nested map.



7. Select the **For Each** transform to open the nested map. Then, define the transformation logic inside the **For Each** nested map.

   For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.

Example

The following example shows how to define the mapping logic in a **For Each** transform after you complete steps 1 to 6:

- Each instance of the **For Each** transform produces an output message.
- The repeating element **H** has three elements that are used to set the values of the output message, which only has two elements.
- An additional input element is needed as part of the transformation. This element is used as part of the conditional expression that determines when a set of transformations is applied.
- An **If** transform is used to model the conditional mapping requirements required to set the value of the output message elements for each index of the **For Each** transform.
- When the value of the input element **A** is equal to the value of the input element **H1**, then the **If** transform is applied. Otherwise, the transformation logic in an **Else** transform is applied.

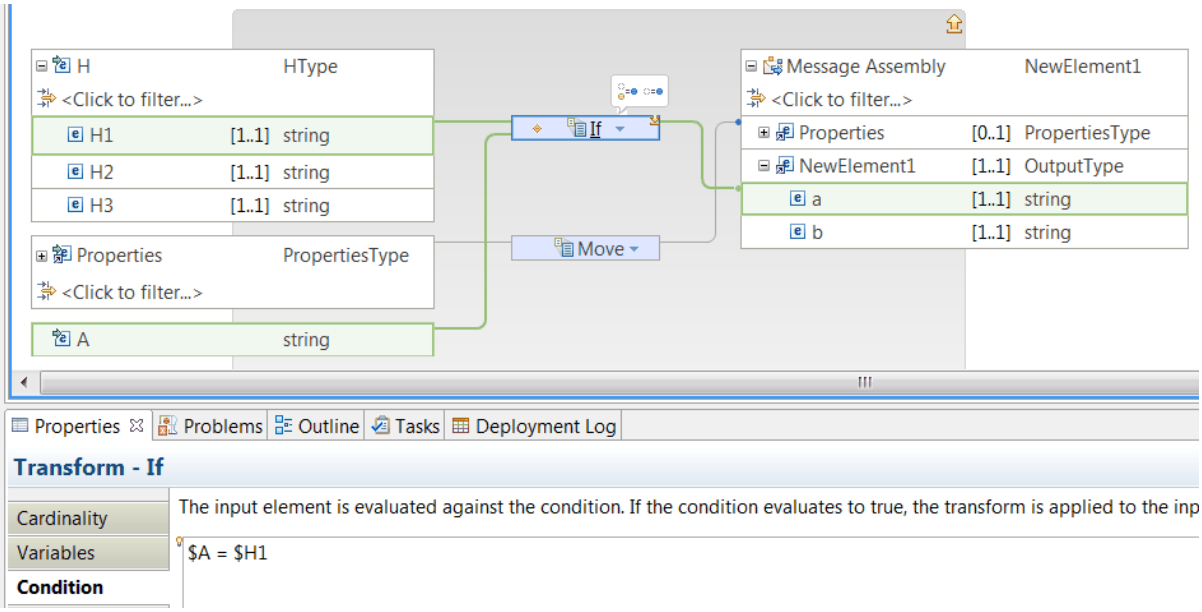
.

Complete the following steps to define the transformation logic:

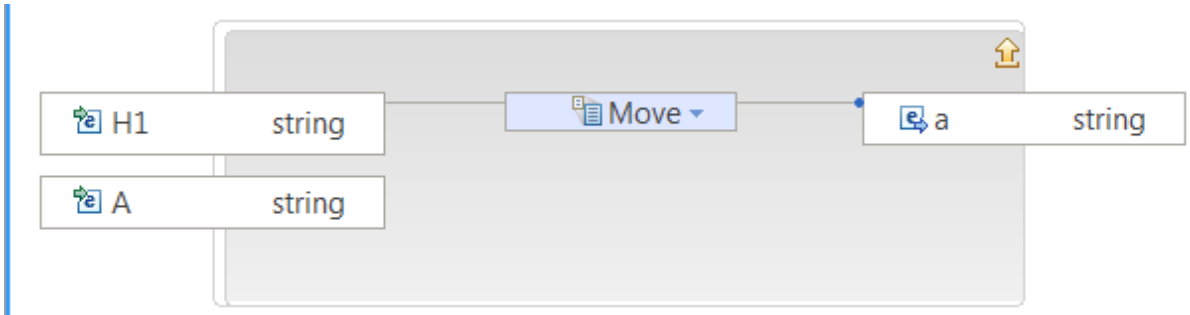
Open the **For Each** transform nested map.

Define an **If** transform between **H** and **a**.

- Define a primary connection between **H** and the **If** transform.
- Define a supplement connection between **A** and the **If** transform.
- Define a connection between the **If** transform and the output element **a**.

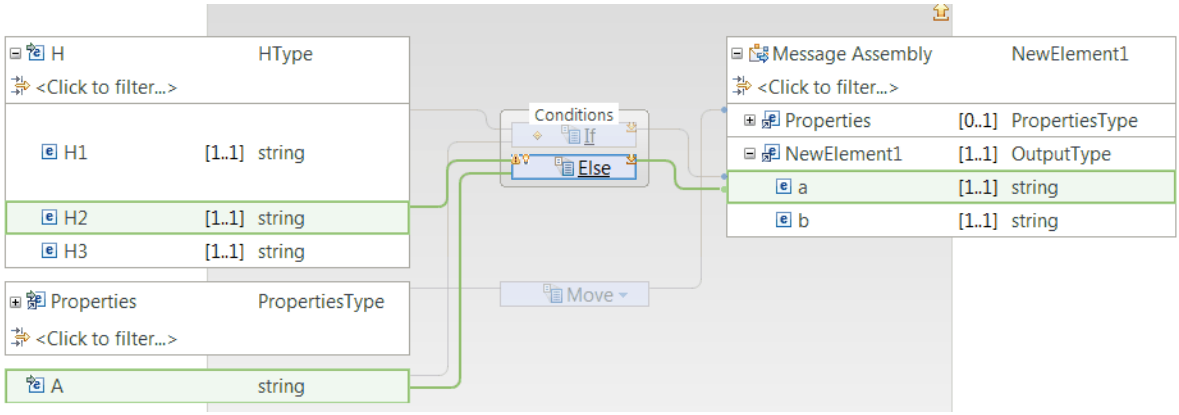


Define a **Move** transform between **H1** and **a**. This operation defines the transformation logic that the **If** transform performs.

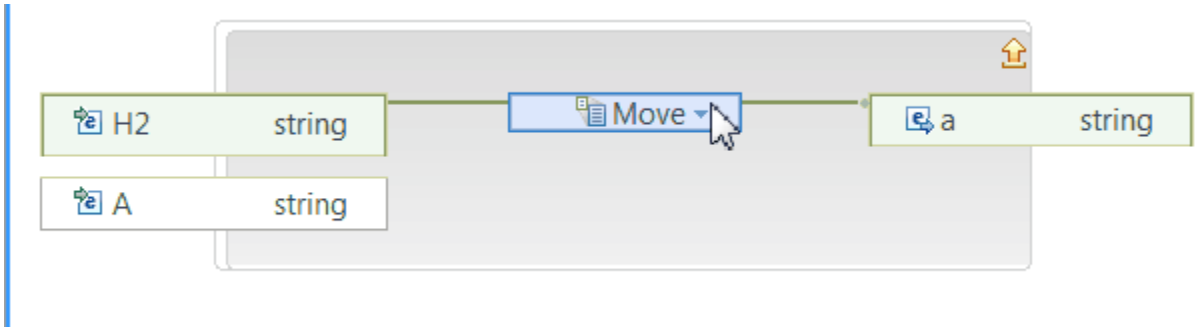


Add the **Else** transform:

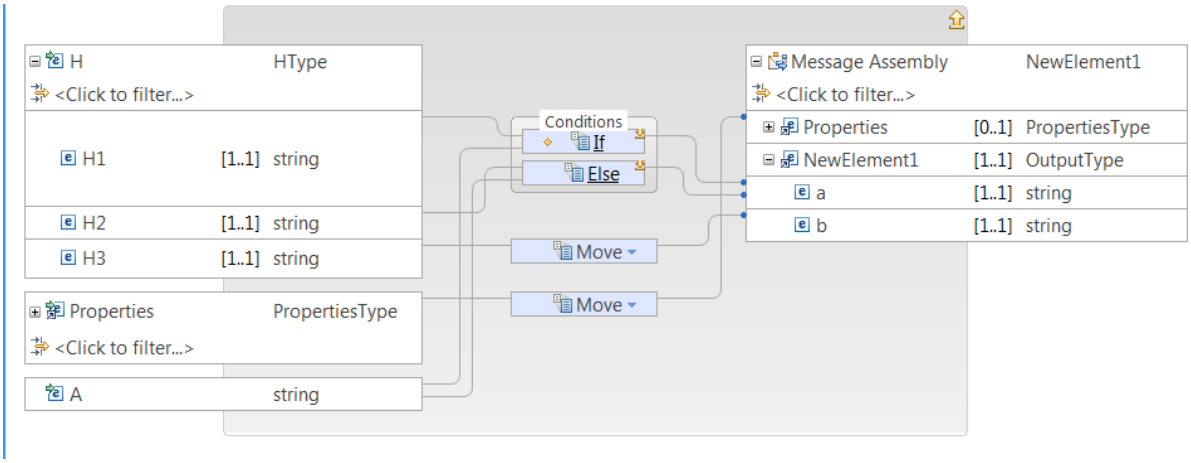
- Define a primary connection between **H** and the **Else** transform.
- Define a supplement connection between **A** and the **Else** transform.
- Define a connection between the **Else** transform and the output element **a**.



Define a **Move** transform between **H2** and **a**. This operation defines the transformation logic that the **Else** transform performs.



The following figure shows the **For Each** transform nested map after all the transformation logic has been implemented:



Mapping an input message into different output messages

You can use the **If** transform to create a map that takes a single input message and produces a different output message based on the conditional expression that you define.

About this task

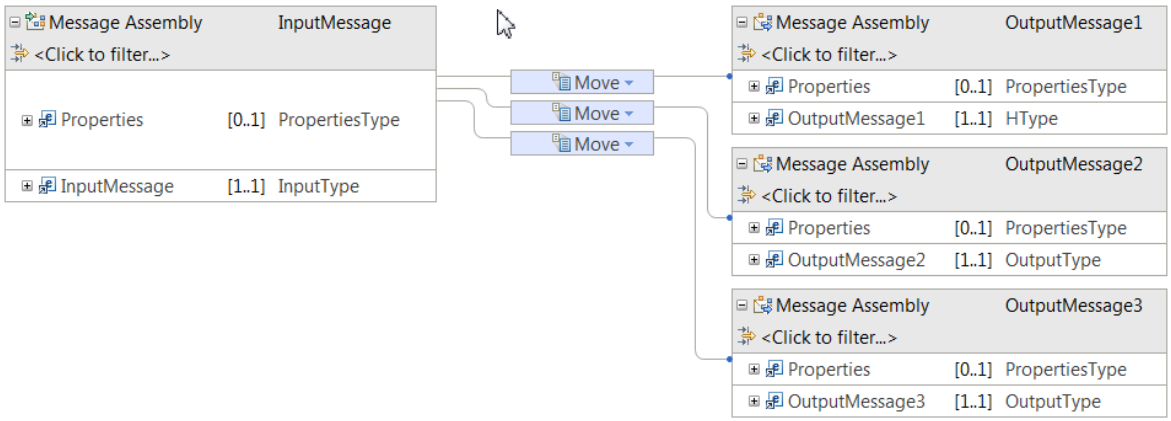
When you configure multiple output message assemblies, each output message assembly has its own properties.

You can configure each output message assembly independently of the others.

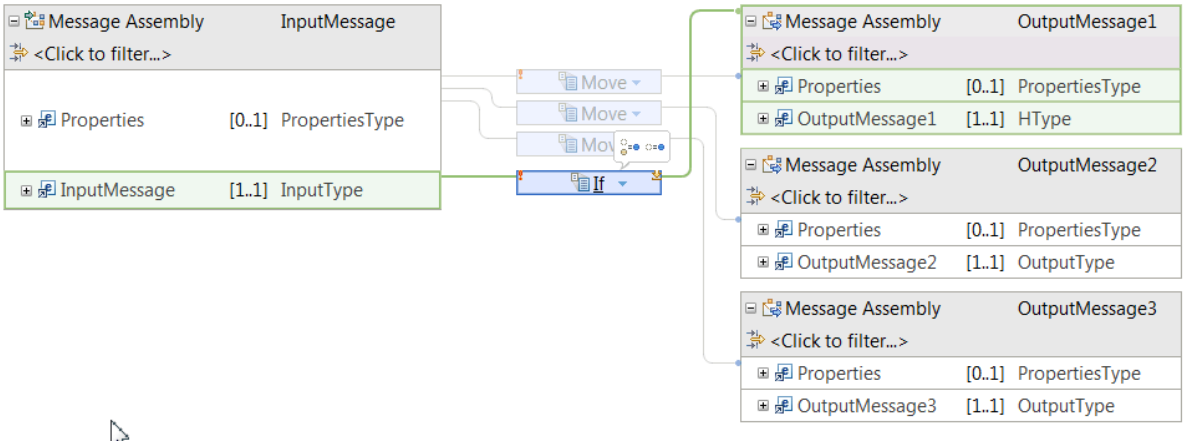
Procedure

Complete the following steps to split a message into different output messages by using the **If** transform:

1. Create a map, and add one input element to your input message assembly, and two or more output message assemblies.

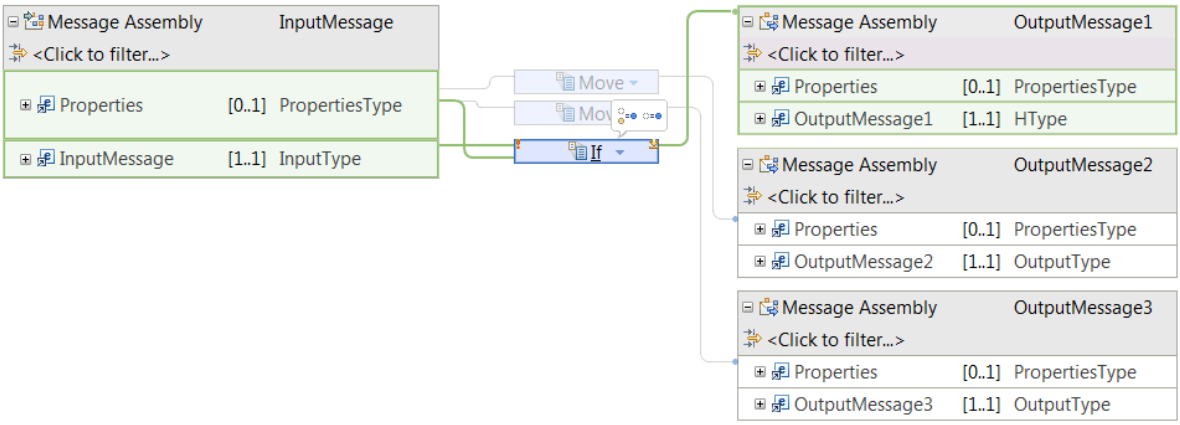


2. Define an **If** transform between the input element and one of the output message assemblies.

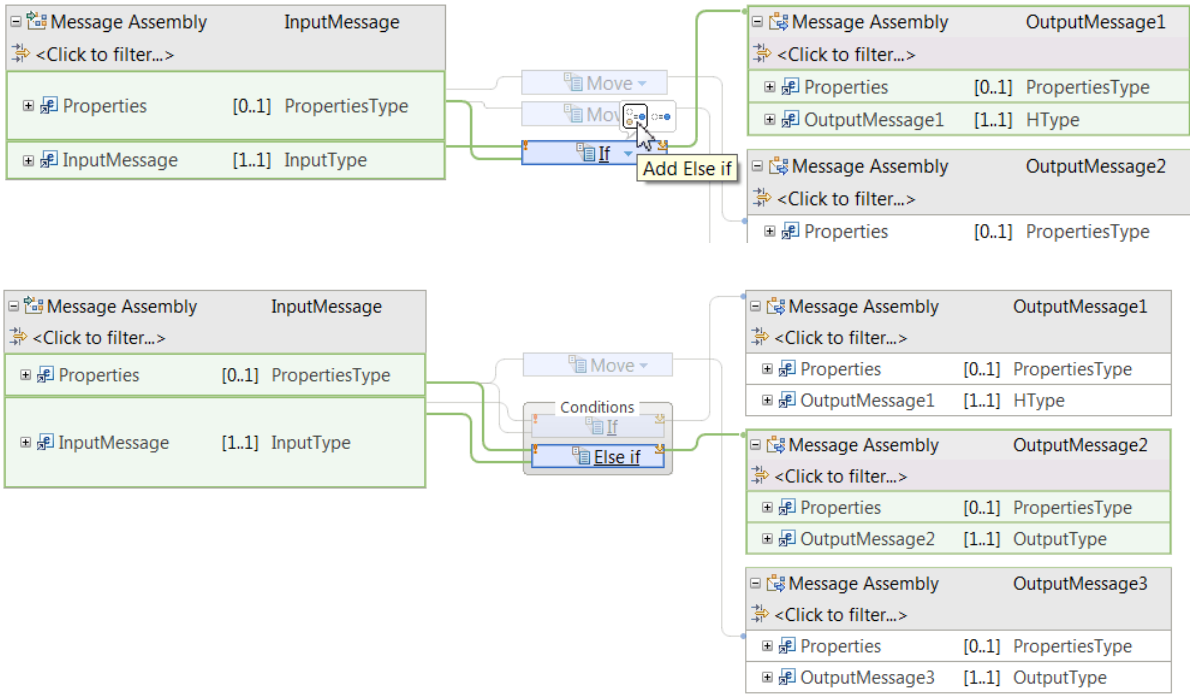


The **Move** transform between the input and the output Properties tree of the first message assembly will display an error. Continue with the steps to remove the error.

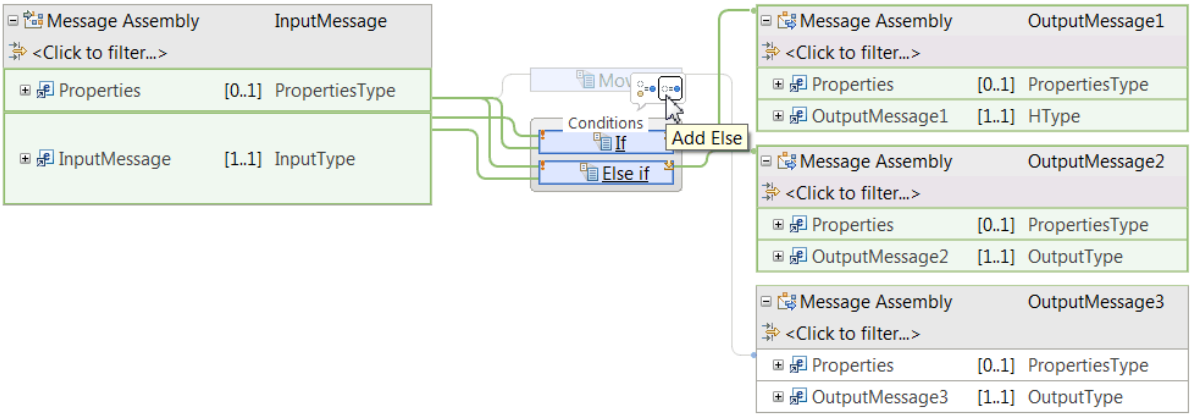
3. Delete the **Move** transform marked with an error, and then connect the input Properties tree to the **If** transform.

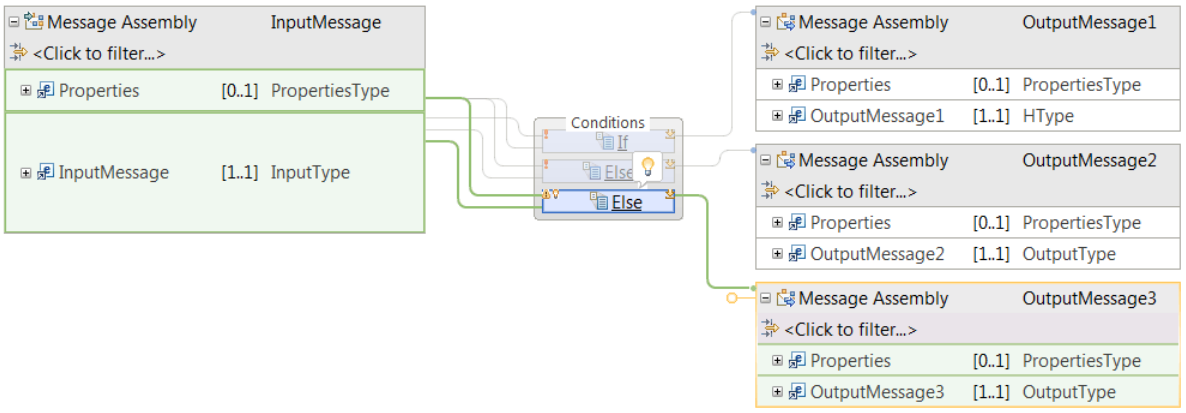


4. Optional: Add the **ElseIf** transform and complete the following steps to connect it to a different output message assembly:
   a) Delete the **Move** transform that connects the Properties tree and the second output message assembly.
   b) Define a connection between the Properties tree and the **ElseIf** transform.
   c) Define a connection between the input message and the **ElseIf** transform.
   d) Define a connection between the **ElseIf** transform and the second output message assembly.

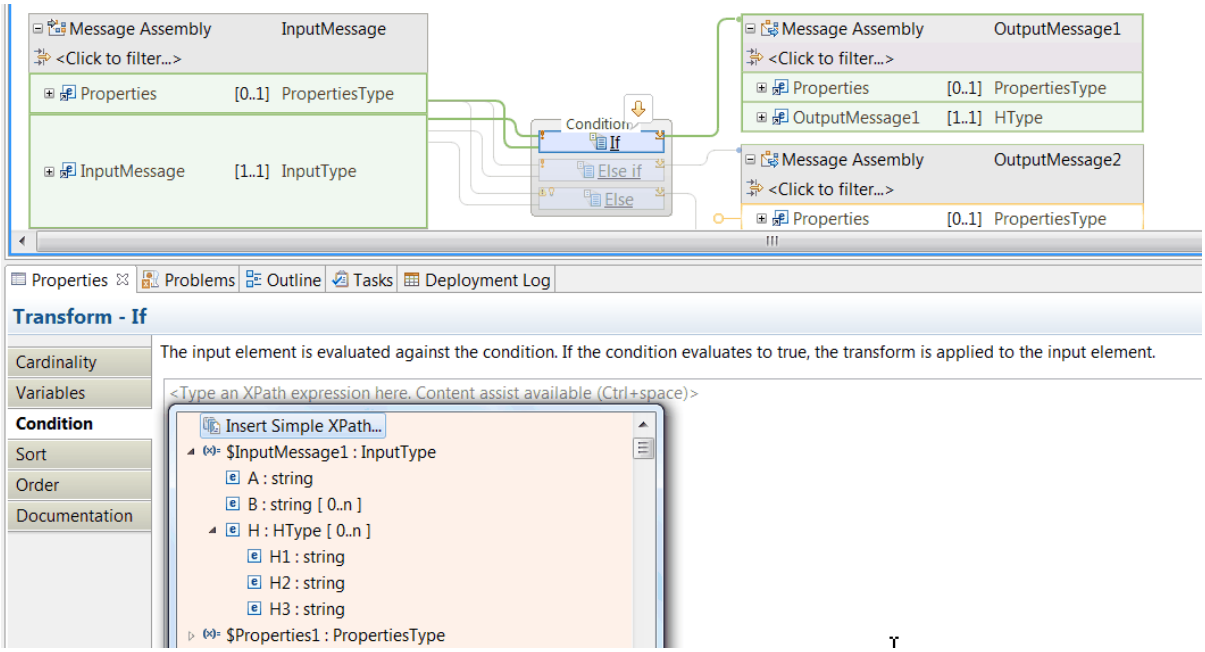


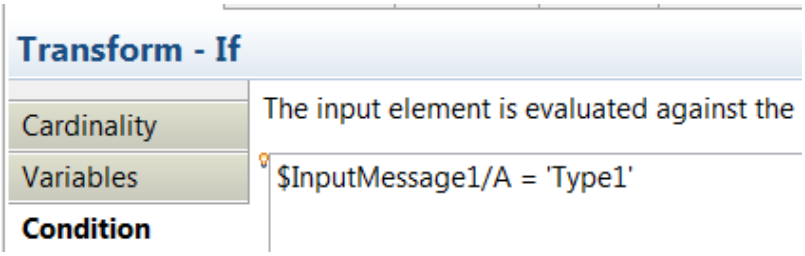
5. Add the **Else** transform and complete the following steps to connect it to a different output message assembly:
   a) Delete the **Move** transform that connects the Properties tree and the third output message assembly.
   b) Define a connection between the Properties tree and the **Else** transform.
   c) Define a connection between the input message and the **Else** transform.
   d) Define a connection between the **Else** transform and the second output message assembly.



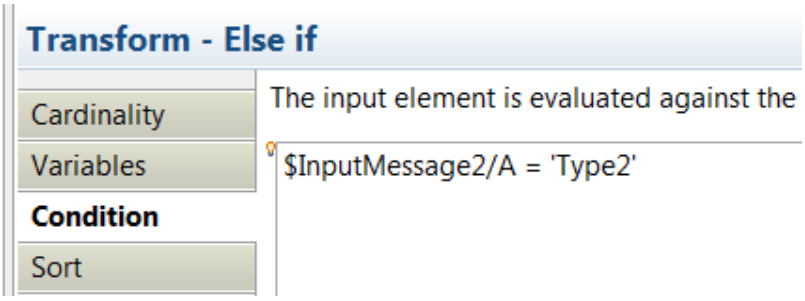


6. Define the conditional expression that determines when the **If** transform is applied and a message based on the first output message assembly is created.
   a) Open the Properties view of the **If** transform.
   b) Define an XPath expression in the **Condition** tab. Use content-assist. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.





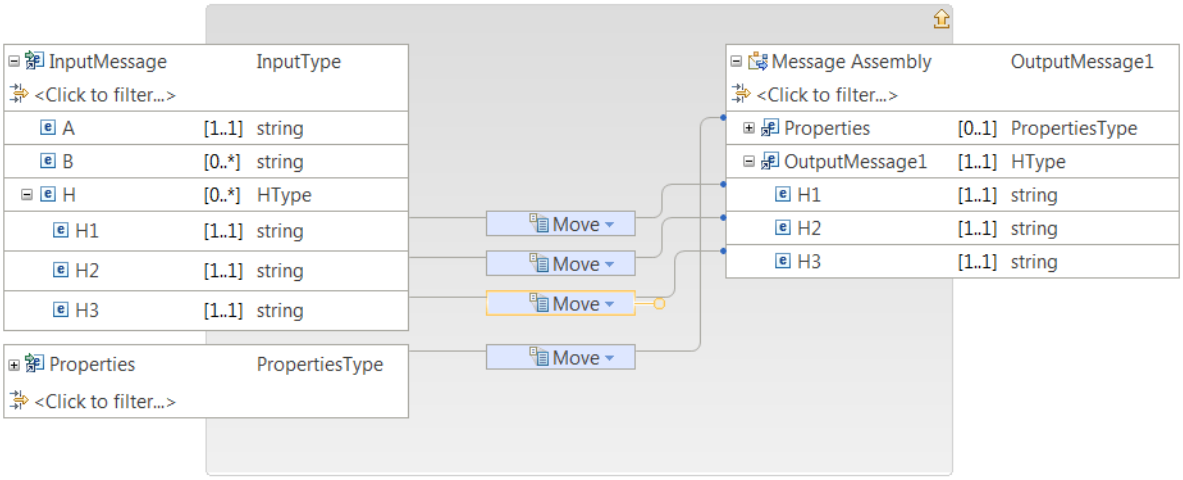
7. Define the conditional expression that determines when the **ElseIf** transform is applied and a message based on the second output message assembly is created.
   a) Open the Properties view of the **If** transform.
   b) Define an XPath expression in the **Condition** tab. Use content-assist. For more information, see "Defining an XPath conditional expression for a transform" on page 1433.



Note: When the conditional expression of the **If** transform and the **ElseIf** transform evaluate to **false**, the transformation logic defined for the **Else** transform is applied.

8. Select the **If** transform to open the associated nested map. Then, define transforms between the input and the output elements. Remember to connect the input Properties to the output properties tree with a **Move** transform.

   By default, a **Submap** transform is defined. You can choose to create a submap with your transformation logic, or delete the **Submap** transform and define locally your transformation logic.



   For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282.
9. Select the **ElseIf** transform to open the associated nested map. Then, define transforms between the input and the output elements.
10. Select the **Else** transform to open the associated nested map. Then, define transforms between the input and the output elements.

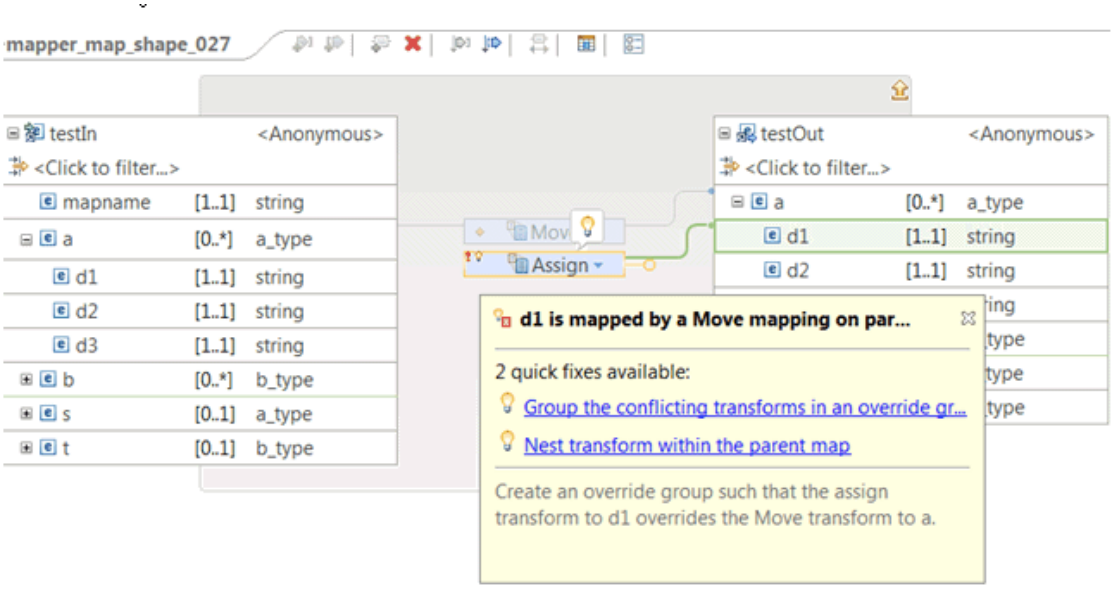
Applying mapping overrides

You can use the **Override** function to copy a complex type from the input to the output object, while updating some of the child elements in the complex type.

About this task

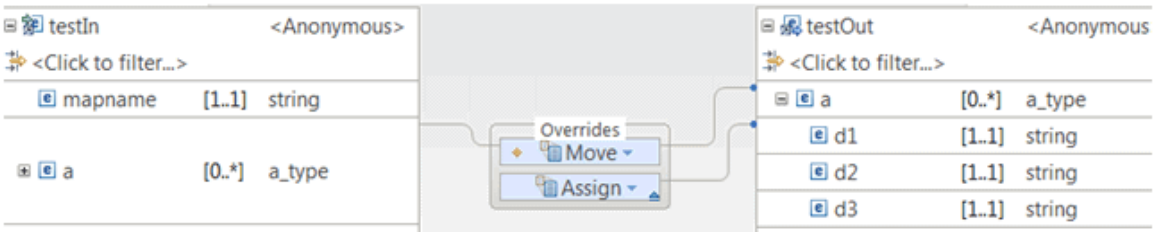
In a message map, you copy a complex type by wiring a **Move** transform from the input object to the output object. You can also add additional transforms to set the value of output elements in the complex object.

As the parent is already mapped, the Graphical Data Mapping editor displays an error marker and offers a Quick Fix.



To resolve the error generated when you copy a complex type while modifying some of the output elements, select the option **Group the conflicting transforms in an overrides group**.

Note: The transforms contained in the overrides group are applied after the parent transform is applied.



Accessing a global cache by using a Mapping node

You can use a Mapping node to interact with a map in an embedded cache or external WebSphere eXtreme Scale grid.

Before you begin

- Read the concept information in Data caching overview.
- Ensure that the embedded cache or external grid is available. The embedded cache is disabled by default. To use the cache, select an integration node level cache policy; see Configuring the embedded global cache.
- If you are connecting to an external grid, create a configurable service to define how the connection is made. For more information, see mqsicreateconfigurableservice command (deprecated).

About this task

You can use a **Cache Put** transform in a Mapping node to store data in the global cache; see “Adding key-value pairs to the global cache by using a Mapping node” on page 1515.

You can use a **Cache Get** transform in a Mapping node to retrieve data from the cache for processing or routing; see “Retrieving a value from the global cache by using a Mapping node” on page 1517.

You can use a **Cache Remove** transform in a Mapping node to remove a key-value pair from the cache; see “Removing a key-value pair from the global cache by using a Mapping node” on page 1519.

If you want to handle any exceptions that are returned by the **Cache** transform, you can add a **Cache Failure** transform to the **Cache** transform group; see "Handling global cache exceptions in a graphical data map" on page 1522.

Adding key-value pairs to the global cache by using a Mapping node
You can use a Mapping node to add key-value pairs to a map in an embedded cache or a WebSphere eXtreme Scale grid.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Creating a message map" on page 1382.

About this task

You can use a **Cache Put** transform in a Mapping node to store data in the global cache.

Interactions with the global cache happen outside the message flow transaction, and are committed immediately. If an error occurs downstream of the node that interacts with the global cache, the cache interactions are not rolled back.

To configure a graphical data map to add data to the global cache, complete the following steps:

Procedure

1. Open the graphical data map in the Graphical Data Mapping editor.
2. In the toolbar, click the **Cache Put** icon ( ) to add a **Cache Put** transform to the canvas.
3. Optional: By default, the **Cache Put** transform adds a new key-value pair to the global cache. You can change the action by selecting one of the following options from the General tab of the Properties for the **Cache Put** transform:
   - **Insert entry into cache** (default): Add the key-value pair to the cache.
   - **Update entry in cache**: Update the value that is associated with the key that is already in the cache.
   - **Insert or update entry in cache**: If the key already exists in the cache, update the value that is associated with the key. If the key does not already exist in cache, add the key-value pair to the cache.
4. Provide values for the parameters in the **Cache Put** transform by completing one of the following steps:
   - Map the input elements to the **Cache Put** transform.

     Note: You can also select the elements before you click the **Cache Put** icon. The elements that you select are then automatically mapped to the **Cache Put** transform.
   - Identify fixed values or user-defined properties to assign to the parameters.

   For example, in the following image, the Value element is mapped to the **Cache Put** transform.

5. Double-click the **Cache Put** transform to edit the nested map.

   To edit the nested map, you can also perform any of the following operations:

   - Click the **Cache Put** link in the **Cache Put** transform.
   - Click the arrow in the upper right corner of the **Cache Put** transform.
   - Place the mouse over the **Cache Put** transform, and click **Click here** in the pop-up window.

   The output elements of the nested map are the predefined parameters that are required to perform the **Cache Put** transform. If you mapped elements to the **Cache Put** transform, the elements are displayed as input elements in the nested map.

6. Required: Provide values for the **Value** and **Key** parameters of the **Cache Put** transform by using one of the following options:

   Note: The **Cache Put** transform is marked with an error until a value is provided for the **Value** and **Key** parameters.

   - Map an element from the input tree to a parameter. When you map an input element to the **Value** parameter, you are prompted to apply a mapping cast to redefine the default `xs:anySimpleType` schema type to a specific schema type (see "Casting elements in a message map" on page 1398), so that the global cache entry is created with a specific Java Object type. If you do not use a mapping cast on the **Value** parameter, the Java Object type that is created in the global cache might vary depending on the type of the mapped input element; see "Java Object types that are created for each schema input type" on page 1295. Alternatively, you can use an `xs:type` transform to cast the mapped input element, or assigned value; see "Cast type (xs:type)" on page 1299.

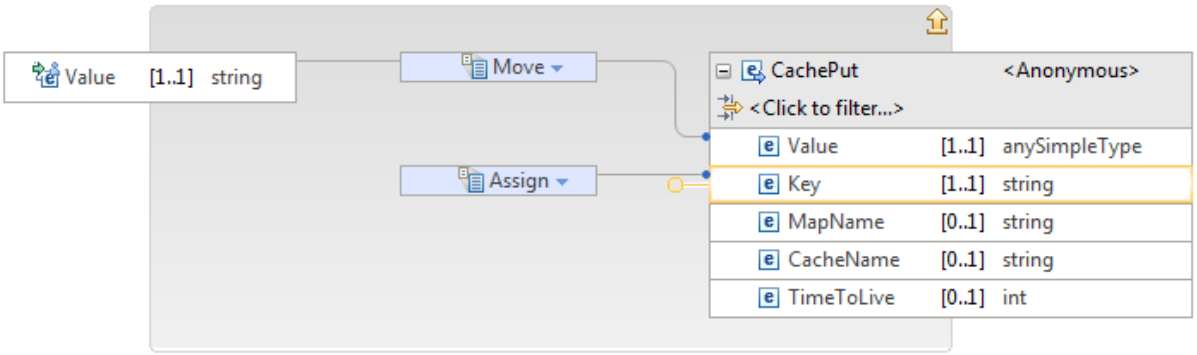
     Note: If the input element is nillable, use one of the following options:

     – Do not cast the target parameter.

     – Make sure that you cast the target parameter to a `String` type, which can have a null value.

   - Set a fixed value for a parameter by right-clicking the parameter in the **Cache Put** transform, clicking **Add Assign**, and adding the required value in the Properties pane. For more information about the **Assign** transform, see "Assign" on page 1289.
   - Set a value for a parameter by using a user-defined property; see "Accessing user-defined properties from a Mapping node" on page 1479.

   For example, in the following image, the content for the **Value** parameter is provided by mapping an element that is named **Value**, and the content for the **Key** parameter is provided by assigning a fixed value.



7. Optional: Map or assign values to the **MapName**, **CacheName**, or **TimeToLive** parameters; for information about valid values for these parameters, see "Cache Put" on page 1293.

8. Optional: Right-click the **Cache Put** transform and select **Cache** > **Return** to implement a nested mapping that is called if the **Cache Put** transform completes successfully. For more information, see "Cache Return" on page 1297.

   Note: You can also add a **Cache Return** transform by placing the mouse over the **Cache Put** transform and selecting the **Cache Return** icon ().
9. Optional: Right-click the **Cache Put** transform and select **Cache** > **Failure** to implement a nested mapping that is called if the **Cache Put** transform fails. For more information, see "Handling global cache exceptions in a graphical data map" on page 1522.

   Note: You can also add a **Cache Failure** transform by placing the mouse over the **Cache Put** transform and selecting the **Cache Failure** icon ().
10. Save the graphical data map.

Results

You added a key-value pair to the global cache.

What to do next

Complete the following tasks:

- Make sure that your global cache is available; see Configuring the embedded global cache or Connecting to a WebSphere eXtreme Scale grid.
- Add the map to a message flow, and deploy the message flow.

For more information about the **Cache Put** transform, see "Cache Put" on page 1293.

Retrieving a value from the global cache by using a Mapping node

You can use a Mapping node to retrieve a value from a map in an embedded cache or a WebSphere eXtreme Scale grid.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Creating a message map" on page 1382.

About this task

You can use a **Cache Get** transform in a Mapping node to retrieve data from the global cache.

To configure a graphical data map to retrieve data from the global cache, complete the following steps:

Procedure

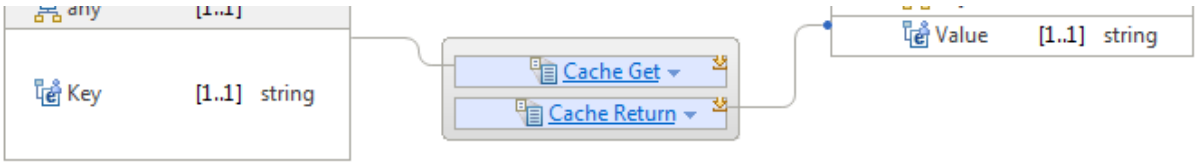
1. Open the graphical data map in the Graphical Data Mapping editor.
2. In the toolbar, click the **Cache Get** icon () to add a **Cache Get** transform to the canvas.
   When you add a **Cache Get** transform, you automatically add a **Cache Return** transform.

3. Provide values for the parameters in the **Cache Put** transform by completing one of the following steps:
   - Map the input elements to the **Cache Get** transform.
   - Identify fixed values or user-defined properties to assign to the parameters.

   Note: You can also map elements from the input tree to the **Cache Return** transform and use these elements to transform the returned cache value.
4. Optional: Map the **Cache Return** transform to the output element that you want to receive the value that is associated with the key in the key-value pair.

   Note: You can also select the elements before you click the **Cache Get** icon. The elements that you select in the input tree are then automatically mapped to the **Cache Get** transform, and the **Cache Return** transform is mapped to the elements that you select in the output tree.

   For example, in the following image, the Key element is mapped to the **Cache Get** transform, and the **Cache Return** transform is mapped to the Value element.



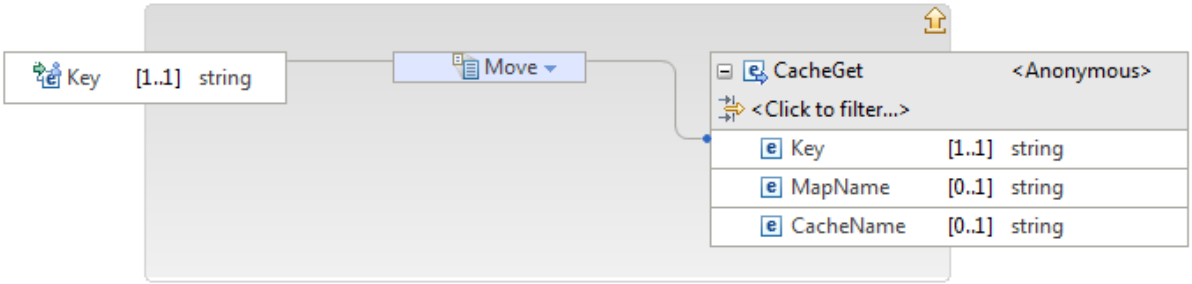
5. Double-click the **Cache Get** transform to edit the nested map.

   Note: To edit the nested map, you can also perform any of the following operations:
   - Click the **Cache Get** link in the **Cache Get** transform.
   - Click the arrow in the upper right corner of the **Cache Get** transform.
   - Place the mouse over the **Cache Get** transform, and click **Click here** in the pop-up window.

   The output elements of the nested map are the predefined parameters that are required to perform the **Cache Get** transform. If you mapped elements to the **Cache Get** transform, the elements are displayed as input elements in the nested map.
6. Required: Provide a value for the **Key** parameter of the **Cache Get** transform by using one of the following options:

   Note: The **Cache Get** transform is marked with an error until a value is provided for the **Key** parameter.
   - Map an element from the input tree.
   - Set a fixed value for the key by right-clicking the **Key** parameter in the **Cache Get** transform, clicking **Add Assign**, and adding the required value in the Properties pane. For more information about the **Assign** transform, see "Assign" on page 1289.
   - Set a value for the key by using a user-defined property; see "Accessing user-defined properties from a Mapping node" on page 1479.

   For example, in the following image, the content for the **Key** parameter is provided by mapping an element that is named **Key** from the input tree.



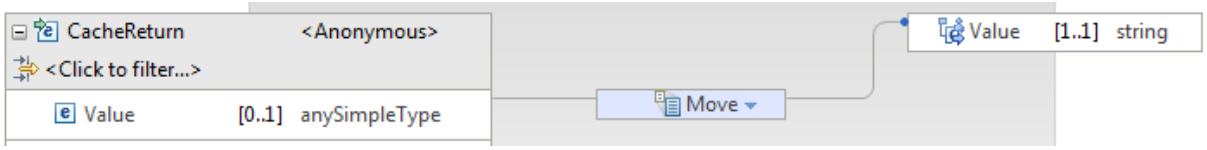
7. Optional: Map or assign values to the **MapName**, or **CacheName** and **MapName** parameters; for information about valid values for these parameters, see "Cache Get" on page 1291.
8. Navigate out of the **Cache Get** transform by clicking the arrow icon ( ).
9. Double-click the **Cache Return** transform to edit the nested map and process the returned value.

   If you mapped the **Cache Return** transform to an element from the output tree, the element is displayed in the nested map.
10. Optional: Map the **Value** element in the **Cache Return** to an element in the output tree.

    The predefined **Value** input element in the **Cache Return** nested mapping has a default schema type of `xs:anySimpleType`. If you know the type of the Java object in the cache, you can map directly to a target element that has the schema type that matches the Java Object type; see "Schema types that are returned for each supported Java Object type" on page 1298.

    If the entry in the cache might have one of several Java Object types, you can apply mapping casts to the predefined **Value** input element. For more information, see "Cache Return" on page 1297.

    For example, in the following image, the **Value** element in the **Cache Return** transform is mapped to the element that is named **Value** in the output tree.



11. Optional: Right-click the **Cache Get** transform and select **Cache** > **Failure** to implement a nested mapping that is called if the **Cache Get** transform fails. For more information, see "Handling global cache exceptions in a graphical data map" on page 1522.

    Note: You can also add a **Cache Failure** transform by placing the mouse over the **Cache Get** transform and selecting the **Cache Failure** icon ( ).
12. Save the graphical data map.

Results

You retrieved a value from the global cache.

What to do next

Complete the following tasks:

- Make sure that your global cache is available; see Configuring the embedded global cache or Connecting to a WebSphere eXtreme Scale grid.
- Add the map to a message flow, and deploy the message flow.

For more information about the **Cache Get** transform, see "Cache Get" on page 1291.

Removing a key-value pair from the global cache by using a Mapping node

You can use a Mapping node to remove a key-value pair from a map in an embedded cache or a WebSphere eXtreme Scale grid.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Creating a message map" on page 1382.

About this task

You can use a **Cache Remove** transform in a Mapping node to remove data from the global cache.

Interactions with the global cache happen outside the message flow transaction, and are committed immediately. If an error occurs downstream of the node that interacts with the global cache, the cache interactions are not rolled back.

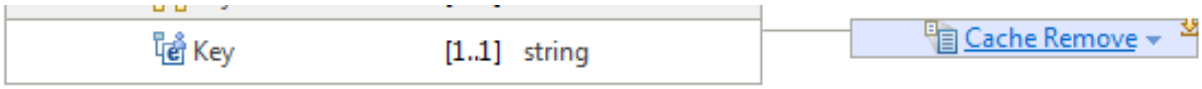
To configure a graphical data map to remove data from the global cache, complete the following steps:

Procedure

1. Open the graphical data map in the Graphical Data Mapping editor.
2. In the toolbar, click the **Cache Remove** icon ( ) to add a **Cache Remove** transform to the canvas.
3. Provide values for the parameters in the **Cache Remove** transform by completing one of the following steps:
   - Map the input elements to the **Cache Remove** transform.

     Note: You can also select the elements before you click the **Cache Remove** icon. The elements that you select are then automatically mapped to the **Cache Remove** transform.
   - Identify fixed values or user-defined properties to assign to the parameters.

   For example, in the following image, the Key element is mapped to the **Cache Remove** transform.

4. Double-click the **Cache Remove** transform to edit the nested map.

   Note: To edit the nested map, you can also perform any of the following operations:

   - Click the **Cache Remove** link in the **Cache Remove** transform.
   - Click the arrow in the upper right corner of the **Cache Remove** transform.
   - Place the mouse over the **Cache Remove** transform, and click **Click here** in the pop-up window.

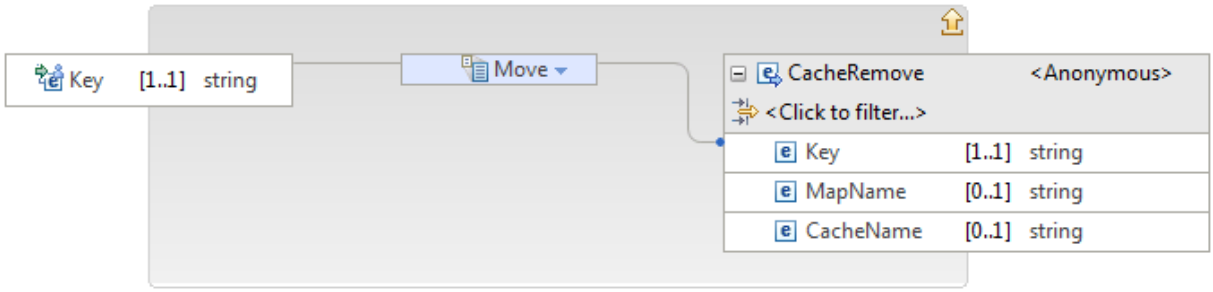
   The output elements of the nested map are the predefined parameters that are required to perform the **Cache Remove** transform. If you mapped elements to the **Cache Remove** transform, the elements are displayed as input elements in the nested map.

5. Required: Provide a value for the **Key** parameter of the **Cache Remove** transform by using one of the following options:

   Note: The **Cache Remove** transform is marked with an error until a value is provided for the **Key** parameter.

   - Map an element from the input tree.
   - Set a fixed value for the key by right-clicking the **Key** parameter in the **Cache Remove** transform, clicking **Add Assign**, and adding the required value in the Properties pane. For more information about the **Assign** transform, see "Assign" on page 1289.
   - Set a value for the key by using a user-defined property; see "Accessing user-defined properties from a Mapping node" on page 1479.

   For example, in the following image, the content for the **Key** parameter is provided by mapping an element that is named **Key** from the input tree.



6. Optional: Map or assign values to the **MapName**, or **CacheName** and **MapName** parameters; for information about valid values for these parameters, see "Cache Remove" on page 1296.
7. Optional: Right-click the **Cache Remove** transform and select **Cache** > **Return** to implement a nested mapping that is called if the **Cache Remove** transform completes successfully.

   Note: You can also add a **Cache Return** transform by placing the mouse over the **Cache Remove** transform and selecting the **Cache Return** icon ( ).

   The nested mapping in the **Cache Remove** transform provides an input element (Value) for the **Cache Return** transform. The input element provides the value of the global cache entry that was removed.

   The predefined **Value** input element in the **Cache Return** nested mapping has a default schema type of `xs:anySimpleType`. If you know the type of the Java object in the cache, you can map directly to a target element that has the schema type that matches the Java Object type; see "Schema types that are returned for each supported Java Object type" on page 1298.

   If the entry in the cache might have one of several Java Object types, you can apply mapping casts to the predefined **Value** input element. For more information, see "Cache Return" on page 1297.

   At run time, if no entry exists in the global cache with the provided key, the **Cache Remove** transform still completes successfully and invokes the **Cache Return** transform but the Value element does not exist.
8. Optional: Right-click the **Cache Remove** transform and select **Cache** > **Failure** to implement a nested mapping that is called if the **Cache Remove** transform fails. For more information, see "Handling global cache exceptions in a graphical data map" on page 1522.

   Note: You can also add a **Cache Failure** transform by placing the mouse over the **Cache Remove** transform and selecting the **Cache Failure** icon ( ).
9. Save the graphical data map.

Results

You removed a key-value pair from the global cache.

What to do next

Complete the following tasks:

- Make sure that your global cache is available; see Configuring the embedded global cache or Connecting to a WebSphere eXtreme Scale grid.
- Add the map to a message flow, and deploy the message flow.

For more information about the **Cache Remove** transform, see "Cache Remove" on page 1296.

Handling global cache exceptions in a graphical data map
Add a **Cache Failure** transform to your **Cache** transform group to handle exceptions that might be raised as a result of a **Cache** transform.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Creating a message map" on page 1382.
- Add a **Cache** transform to the graphical data map; see "Accessing a global cache by using a Mapping node" on page 1514.

About this task

If you want a graphical data map to handle exceptions that are returned as a result of a **Cache** transform, instead of stopping the map, you can add a **Cache Failure** transform to the **Cache** transform group. The **Cache Failure** transform is an optional transform in each of the **Cache** transform groups, and can be added or removed as required. If an exception is raised by the **Cache** transform, and you did not add a corresponding **Cache Failure** transform, the map operation is stopped and the exception is passed by the Mapping node to the message flow, where it can be caught by a **TryCatch** node or passed to the input node.

To add a **Cache Failure** transform to a **Cache** transform group by using the Graphical Data Mapping editor, complete the following steps:

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click a **Cache Put**, **Cache Get**, or **Cache Remove** transform, and then select **Cache** > **Failure**.

   Note: You can also add a **Cache Failure** transform by placing the mouse over the **Cache** transform and selecting the **Cache Failure** icon ().

   A **Cache Failure** transform is added to your **Cache** transform group.
2. Connect the **Cache Failure** transform to specify how any exceptions from the **Cache** transform are processed when the map is run. If the **Cache Failure** transform is present and is connected to one or more output objects, then the exception is caught and processed by the **Cache Failure** transform.

   Important: If the **Cache Failure** transform is present but is not connected, the following actions are taken:

   - The exception is caught by the **Cache Failure** transform and is ignored.
   - The map is marked with a warning.
3. Double-click the **Cache Failure** transform to open the nested map and further define the transform. For more information, see "Cache Failure" on page 1290.

Results

You added a **Cache Failure** transform to a **Cache** transform group in your graphical data map and configured the **Cache Failure** transform to handle exceptions that are returned by the **Cache** transform. If you want the execution of the map to stop when the **Cache** transform receives an exception, remove the **Cache Failure** transform from the **Cache** transform group.

Mapping database content

Use a message map to map or modify database content.

About this task

In the Graphical Data Mapping editor, you can access a database and manipulate your business data.

- You can use database content as input data for a graphical data map transform.
- You can use database transforms to modify database content.
- You can use a database transform to invoke a database stored procedure.

During the development phase, you must configure the database before you can access the data from a message flow in the IBM App Connect Enterprise Toolkit. IBM App Connect Enterprise supports the databases that are listed in IBM App Connect Enterprise Requirements.

For each database transform in your message map, the Graphical Data Mapping editor uses a database definition file (`.dbm` file) to determine the name and structure of the database that you want to access.

When your message map has been deployed, the IBM App Connect Enterprise runtime component connects to the database that is used by transforms in a map, by using a JDBC Providers Policy. The name of the JDBC Providers Policy must match the data source that was used in the Toolkit when the map was created (the discovered database name). You must also ensure that the JDBC Providers Policy is placed in the configured default policy project.

IBM App Connect Enterprise can access databases that are set up on the local computer or on a remote server, subject to restrictions. For more information, see IBM App Connect Enterprise Requirements.

The following topics guide you through the steps that are involved in mapping database content:

- “Creating a database definition (.dbm file) by using the New Database Definition File wizard” on page 1475
- “Selecting data from a table” on page 1523
- “Modifying data in a database by using mapping ” on page 1525
- “Calling a stored procedure from a map” on page 1532

Optionally, you can use JDBC connection pooling with your JDBC Providers policy to manage your database resources. For more information, see JDBC Providers policy.

Selecting data from a table

To map an output element from a database table, use the Graphical Data Mapping editor to retrieve the relevant rows from the database and then populate the output elements with values from the database.

Before you begin

You must complete the following tasks:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see “Creating a message map” on page 1382.

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, click the **Select rows from a database** icon.

    .If you include a Select transform within a ForEach nested transform, the IBM App Connect Enterprise runtime component issues one SQL select to the database for each iteration of the ForEach transform.
2. In the **"New database select"** wizard, select the database, table, and column from which you want to select data. To add a database definition file, or to discover a new database by connecting to a database server, click **Add database...**. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.
3. In the **SQL where clause** field, use supported SQL to specify the criteria for selecting the rows from the selected column of your database table.

   Build a supported SQL statement by dragging items from the **Table columns** and **Operators** panes to the **SQL where clause** field.

   To include values in your SQL statement, drag items from the **Available inputs for column values** pane to the **SQL where clause** to add them as parameters, or type literal values such as `'abc'` or 123 directly in the **SQL where clause**.

   Parameters from the **SQL where clause** are listed in the XPath expression table. You can edit the XPath expressions to refine the input, for example to add a specific array index for a dragged repeating field. A default **SQL where clause** is created for you, which selects all rows in your selected database table.

   Note: If you edit the text of the **SQL where clause** directly:

   - Ensure that the case of your table and column names match that of your database.
   - Avoid the use of double-quotes around table and column names.
   - Use only the supported SQL keywords that are presented in the Operators pane.
   - Ensure that each parameter placeholder is inserted as a question mark followed by an optional unique number and a space character, and also ensure that each parameter placeholder is defined with an XPath expression in the parameter table below the SQL where clause.
4. Optional: Select **Treat warning as error**.

   If this option is selected, the first SQL operation that results in a warning from the selected database raises an exception.

   Important: Database warnings are vendor-specific. For more information about database warnings, see the documentation for your database product.
5. Click **OK**.

   A **Select** transform, is created, and the data that you selected is displayed in the graphical data map.
6. Connect the **Select** transform to the required output object in the map.

   The **ResultSet** input to the **Select** transform is a repeating structure that contains one instance for each row that is selected by your configured **SQL where clause**.
7. Click the **Select** transform to further define the transform.

   A nested map is created, in which you can select the specific transforms that are required for the input and output elements.

What to do next

- If you want exceptions that are returned from the database server when the SQL operation is run to be handled by the map, instead of having such exceptions stop the map and being reported, you can add a Failure transform into the transform group; see "Handling database exceptions in a graphical data map" on page 1535.

Modifying data in a database by using mapping
Use the Graphical Data Mapping editor to insert, update, or delete rows of data in a database table.

About this task

You can use database transforms in your graphical data maps to insert new rows of data, or to update or delete existing rows of data, in your database tables. For each database transform in your graphical data map, the Graphical Data Mapping editor uses a database definition file (`.dbm` file) to determine the name and structure of the database that you want to access. You can start the wizard to create a database definition file when you create a database transform in a graphical data map.

If you connect elements from an input object to database columns within a database transform in your graphical data map, every input message that is processed by your map at run time must include those elements. If a message is either missing an element that is connected to a database column, or does not provide a valid value for that database column, an exception is raised when the message is processed by the map. Input elements that you connect to nullable database columns must provide either a valid value, or the NULL value. For more information about null values, see XMLNSC empty elements and null values.

When you add a database Insert, Update or Delete transform to a graphical data map, the transform is displayed as an additional output target to which you can connect input objects. When your map is run, a database transform calls a single operation on the configured database server. If you connect a repeating input element to the database transform, the Graphical Data Mapping editor moves the database transform inside a nested "For Each" transform from the repeating input.

A database Insert, Update, or Delete transform are created as a transform group, comprising the database operation and a Return transform. The database operation transform for Insert and Update are nested transforms in which the individual mapping to the database table columns are made. The Return transform is an optional transform that allows a nested mapping to be entered if the database operation is successful. If you do not want to use the Return transform, you can delete it from the transform group. If you must provide some mapping for when a failure is returned from the database operation, you can add a Failure transform into the transform group. The Failure transform provides a nested transform that is entered if the database system returns a failure.

If the insert, update, or delete is conditional on a test result, you can change the Insert, Update, or Delete transform to an If transform. Before you change the transform, ensure that the Insert, Update, or Delete transform is not part of a transform group. Remove any Return or Failure transform then select an If transform in place of the Insert Update or Delete. The Insert, Update, or Delete transform is moved into the nested mapping of the If transform. You can then add any required Return and Failure transforms.

The following topics describe how to modify data in a database table:

- “Inserting data into a table” on page 1525
- “Updating data in a table” on page 1527
- “Deleting data from a table” on page 1528

Inserting data into a table
Use the Graphical Data Mapping editor to insert data into a database table.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see “Creating a message map” on page 1382.

About this task

To insert a row, or multiple rows, into a database table by using the Graphical Data Mapping editor, complete the following steps:

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click the canvas, and select **Database** > **Insert into table**. Alternatively, select a schema element as an input object, and then click the **Insert a row into a database table** icon.

   The **Insert** wizard is displayed.
2. In the **Database** field, select the database that you want to modify. To add a database definition file, or to discover a new database by connecting to a database server, click **Add database...**. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.

   To use a different database name at run time, you can override this value by setting the **Name of the database** property of the JDBC Providers policy that connects to your database; see JDBC Providers policy (JDBCProviders).
3. In the **Schema** field, select the database schema that you want to use to build the transform.

   To use a different database schema at run time, you can override this value by setting the **Name of the database schema** property of the JDBC Providers policy that connects to your database; see JDBC Providers policy (JDBCProviders).
4. In the **Table** field, select the table that you want to modify.
5. Optional: Select **Treat warning as error**.

   If this option is selected, the first SQL operation that results in a warning from the selected database raises an exception.

   Important: Database warnings are vendor-specific. For more information about database warnings, see the documentation for your database product.
6. Click **OK**.

   An **Insert** transform and a **Return** transform are created as a transform group, and are displayed in your graphical data map. The **Return** transform is an optional transform type. If you do not need to use the **Return** transform, you can delete it from your graphical data map.
7. Optional: To replace a **Return** transform that you deleted from your graphical data map, right-click your **Insert** transform and select **Database** > **Utilize return**.
8. In the Graphical Data Mapping editor, connect input objects to the **Insert** transform to define the content of your inserted row.

   - Connect a non-repeating element to the **Insert** transform to insert a single row into the selected database table.
   - Connect one or more repeating elements to the **Insert** transform to insert multiple rows into the selected database table. To connect multiple repeating elements, select your repeating elements, then right-click the **Insert** transform and select **Add Connection**.

   If you connect a single repeating element, the **Insert** transform is nested inside a **For Each** transform. If you connect multiple repeating elements, the **Insert** transform is nested inside a **Join** transform. In either case, the nested transform opens so you can continue to edit your **Insert** transform.
9. Click the **Insert** transform to create connections to the columns in your inserted row, and to further define the transform.
10. Optional: If you need to provide handling for the connected source element being Missing, Empty or Nil, you can set a Database Policy. See "Behavior when modifying database column values from optional source elements" on page 1535.
11. Optional: Connect the **Return** transform to implement a nested mapping that is called if the **Insert** operation was completed successfully.
12. Optional: Click the **Return** transform to further define the transform.

    A nested map is created, in which you can select the specific transforms that are required for the input and output elements.

What to do next

- If you want exceptions that are returned from the database server when the SQL operation is run to be handled by the map, instead of having such exceptions stop the map and being reported, you can add a Failure transform into the transform group; see "Handling database exceptions in a graphical data map" on page 1535.
- Set up a JDBC connection to the database that you want to access.

Updating data in a table
Use the Graphical Data Mapping editor to update data in a database table.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Graphical Data Mapping editor" on page 1266.

About this task

To update a row of data, or multiple rows of data, in a database table by using the Graphical Data Mapping editor, complete the following steps:

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click the canvas, and select **Database** > **Update Table**. Alternatively, click the **Update a row in a database table** icon.

   The **New Database Table Update** wizard is displayed.
2. In the **Database** field, select the database that you want to modify. To add a database definition file, or to discover a new database by connecting to a database server, click **Add database...**. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.
3. In the **Schema** field, select the database schema that you want to use to build the transform.
4. In the **Table** field, select the database table that you want to modify.
5. Optional: Select **Treat warning as error**.

   If this option is selected, the first SQL operation that results in a warning from the selected database raises an exception.

   Important: Database warnings are vendor-specific. For more information about database warnings, see the documentation for your database product.
6. In the **SQL where clause** field, use supported SQL to specify the criteria for selecting rows from your database table.

   Build a supported SQL statement by dragging items from the **Table columns** and **Operators** panes to the **SQL where clause** field.

   To include values in your SQL statement, drag items from the **Available inputs for column values** pane to the **SQL where clause** to add them as parameters, or type literal values such as `'abc'` or 123 directly in the **SQL where clause**.

   Parameters from the **SQL where clause** are listed in the XPath expression table. You can edit the XPath expressions to refine the input, for example to add a specific array index for a dragged

repeating field. A default **SQL where clause** is created for you, which selects all rows in your selected database table.

Note: If you edit the text of the **SQL where clause** directly, take care to:

- ensure the case of your table and column names match that of your database.
- avoid the use of double-quotes around table and column names.
- only use the supported SQL keywords that are presented in the Operators pane.

7. Optional: Select **Insert when a row does not exist** if you want to insert a new row in your database table when no existing row meets the criteria of your **SQL where clause**.

   If this option is selected, the map checks the "number of rows updated" return from the database server for the Update SQL operation. If the "number of rows updated" is zero, the map issues an insert SQL operation. For the insert operation to succeed, your **Update** transform must explicitly provide valid values for all mandatory database columns. If you want a row that is inserted in this way to use different values to those that are provided by your **Update** transform, consider adding a conditional **Insert** transform inside the **Return** transform.

8. Click **OK**.

   An **Update** transform and a **Return** transform are created as a transform group, and are displayed in your graphical data map. The **Return** transform is an optional transform type that provides a nested mapping that is entered only if the associated **Update** was successful. If you do not need to use the **Return** you can delete it from your graphical data map.

9. Optional: To replace a **Return** transform that you deleted from your graphical data map, right-click your **Update** transform and select **Database** > **Utilize return**.

10. In the Graphical Data Mapping editor, connect input objects to the **Update** transform to define the content of your updated row.

    If you connect a repeating element, the **Update** transform is nested inside a **For Each** transform, and this nested transform is opened so that you can continue to edit your **Update** transform.

11. Click the **Update** transform to create connections to the columns in your updated row, and to further define the transform.

12. Optional: If you need to provide handling for the connected source element being Missing, Empty or Nil, you can set a Database Policy. See “Behavior when modifying database column values from optional source elements” on page 1535.

13. Optional: Connect the **Return** transform to implement a nested mapping that is called if the **Update** operation was completed successfully.

    The nested **Return** transform provides a built-in input, "NumberOfRowsUpdated", and additional inputs can be connected.

14. Optional: Click the **Return** transform to further define the transform.

    A nested map is created, in which you can select the specific transforms that are required for the input and output elements.

What to do next

- If you want exceptions that are returned from the database server when the SQL operation is run to be handled by the map, instead of having such exceptions stop the map and being reported, you can add a Failure transform into the transform group; see “Handling database exceptions in a graphical data map” on page 1535.

Deleting data from a table
Use the Graphical Data Mapping editor to delete data from a database table.

Before you begin

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see "Creating a message map" on page 1382.

About this task

To delete a row of data, or multiple rows of data, from a database table by using the Graphical Data Mapping editor, complete the following steps:

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click the canvas, and select **Database** > **Delete from Table**. Alternatively, click the **Delete a row from a database table** icon.

   The **New Database Table Delete From** wizard is displayed.
2. In the **Database** field, select the database that you want to modify. To add a database definition file, or to discover a new database by connecting to a database server, click **Add database...**. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.
3. In the **Schema** field, select the database schema that you want to use to build the transform.
4. In the **Table** field, select the database table that you want to modify.
5. Optional: Select **Treat warning as error**.

   If this option is selected, the first SQL operation that results in a warning from the selected database raises an exception.

   Important: Database warnings are vendor-specific. For more information about database warnings, see the documentation for your database product.
6. In the **SQL where clause** field, use supported SQL to specify the criteria for selecting the rows that you want to delete from your database table.

   Build a supported SQL statement by dragging items from the **Table columns** and **Operators** panes to the **SQL where clause** field.

   To include values in your SQL statement, drag items from the **Available inputs for column values** pane to the **SQL where clause** to add them as parameters, or type literal values such as `'abc'` or 123 directly in the **SQL where clause**.

   Parameters from the **SQL where clause** are listed in the XPath expression table. You can edit the XPath expressions to refine the input, for example to add a specific array index for a dragged repeating field. A default **SQL where clause** is created for you, which selects all rows in your selected database table.

   Note: If you edit the text of the **SQL where clause** directly:

   - Ensure that the case of your table and column names match that of your database.
   - Avoid the use of double-quotes around table and column names.
   - Use only the supported SQL keywords that are presented in the Operators pane.
   - Ensure that each parameter placeholder is inserted as a question mark followed by an optional unique number and a space character, and also ensure that each parameter placeholder is defined with an XPath expression in the parameter table below the SQL where clause.
7. Click **OK**.

   A **Delete** transform and a **Return** transform are created as a transform group, and are displayed in your graphical data map. The **Return** transform is an optional transform that provides a nested mapping. It is entered only if the associated **Delete** was successful. If you do not need to use the **Return** transform, you can delete it from your graphical data map.

8. Optional: To replace a **Return** transform that you deleted from your graphical data map, right-click your **Insert** transform and select **Database** > **Utilize return**.
9. Optional: Connect the **Return** transform to implement a nested mapping that is called if the **Delete** operation was completed successfully.
10. Optional: Click the **Return** transform to further define the transform.

    A nested map is created, in which you can select the specific transforms that are required for the input and output elements.

What to do next

- If you want exceptions that are returned from the database server when the SQL operation is run to be handled by the map, instead of having such exceptions stop the map and being reported, you can add a Failure transform into the transform group; see "Handling database exceptions in a graphical data map" on page 1535.

Data type considerations for mapping database content
Data type handling using the Graphical Data Mapping editor to read or modify data in a database table requires consideration of the type of Database server that will be connected to from the run time. The map may require to make explicit type casts, in order to avoid mapping node exceptions or database server exceptions being thrown.

IBM App Connect Enterprise can access databases that are set up on the local computer or on a remote server, subject to restrictions.

IBM App Connect Enterprise supports the databases that are listed in IBM App Connect Enterprise Requirements (SOE).

Data types considerations for mapping database content during development

During the development phase, you must configure a database before you can access the data in a map.

To configure a database, you define a database definition file. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475.

A database definition file holds the physical data model that details all the database resources, such as the schema, the tables, and other resources, that you need access to.

Note: A database definition file in the IBM® Integration Toolkit is not automatically updated. If you modify your database, you must re-create the database definition file describing the connection to the database.

For each database transform in your message map, the Graphical Data Mapping editor uses the database definition file (`.dbm` file) to determine the name and structure of the database that you want to access.

When you map a database table in a message map, the data types of the database columns are provided by the database definition file.

You can use the **xs:type** cast transform or custom transforms, such as **Custom XPath**, to ensure that data from elements mapped to the database columns are of the correct type.

Data types considerations for mapping database content at run time

At run time, a JDBC Providers policy is used to determine the database to connect to. You must enable the JDBC connection to the database before you execute a map that requires data from a database. See JDBC Providers policy.

Note: The name of the JDBCProvider service must be identical to the name of the database.

When you use a Mapping node that includes a map with database transforms, you must consider the database server behavior:

- If your database server can provide at run time table parameter meta data, the Mapping node validates the input data to a database transform in a map and performs any allowed implicit type casting, before sending the SQL statement to the database for execution.

  Note: The Mapping node applies any required **xs:type** cast transforms before passing data in SQL statements. If there is no valid type cast between the type of the presented value and the type defined by the database meta data, a **run time exception** is thrown by the Mapping node that is executing the map.
- If your database server cannot provide at run time table parameter meta data, during development, you must define the **xs:type** cast transform to ensure that all data values that will be passed to the database, as parameters to Where clauses or to populate a column in a table, match the data type requirements of the database server.

  Note: Not all database servers supported by IBM App Connect Enterprise provide querying of table meta data in a way IBM App Connect Enterprise can currently process. IBM App Connect Enterprise cannot currently obtain table meta data when connected to the following database server types:

  – Microsoft_SQL_Server
  – Oracle
  – Sybase_JConnect6_05
  – solidDB

  When a database system cannot provide table meta data at run time, the Mapping node cannot perform validation and implicit type casting. The data element values are passed to the database server in the type they are presented, without any casting being performed. This can result in the database system rejecting the value and throwing a **database exception**.

Behavior when a database server can provide at run time table meta data information

The resulting data type of values that are set to the database systems are determined depending on how the input element is wired to the database transform:

- Column values set via **Move** transforms from an message tree element are passed as its given type when it is a base SQL type. For example: Integer, otherwise as character string formatted as per the IBM App Connect Enterprise `getValueAsString()` MbElement method.
- Column values set via **Custom XPath**, **Custom Java** or **Custom ESQL** functions are passed as the type returned by the function.
- Column values set via **Assign** transform will always be passed as character string. If you require a specific type to be assigned, you must use a **Cast** transform of the appropriate `xs` type constructor. For example, to assign the value 1 to an Integer type column, use the `xs:int()` **Cast** transform and set a value of '1' instead of an **Assign** transform.

When using values in **Where** clauses for **Select**, **Update** and **Delete**, the types are determined as follows:

- Literal values are typed according to standard SQL syntax, such as quote character strings, unquoted numbers and so on.
- Values set via XPath expressions to a message tree element are passed as its given type when it is a base SQL type. For example: Integer, otherwise as character string formatted as per the IBM App Connect Enterprise `getValueAsString()` MbElement method.

Behavior when a database server cannot provide at run time table meta data information

When a database system cannot provide table meta data at run time, the Mapping node cannot perform validation and implicit type casting. The data element values are passed to the database server in the type they are presented, without any casting being performed. This can result in the database system

rejecting the value and throwing a **database exception**. The database server raises a SQL invalid type exception.

To resolve this error, you must manually add explicit type casting in the map. Use the **xs:type** transform in the XPath expression of the **Where clause** when you set a value in a target database column or when you pass a value for a stored procedure parameter.

Working with database servers that are not listed in the product SOE for IBM Integration Bus Version 10

This section only applies if your database server is not listed under IBM App Connect Enterprise Requirements (SOE)

By default, the Mapping node queries table meta data by calling **java.sql.PreparedStatement.getParameterMetaData()**. If the database server you are connecting to does not fully implement this JDBC interface method, the mapping can fail due to an **SQL exception** from the database server. For example, the database server might respond by returning the exception java.sql.SQLFeatureNotSupportedException.

You can set the environment variable **MQSI_MAP_DB_PARAMETERMETADATA_SUPPORT** to control whether the Mapping node queries the database server for table meta data at run time.

To set the environment variable, complete the following steps:

1. Create a file in the following directory:
   - On Windows: *work_path*`\Common\profiles`
   - On Linux and UNIX: *work_path*`/common/profiles`

   where *work_path* is the machine-wide IBM App Connect Enterprise working directory.

   Note: To verify the machine-wide IBM App Connect Enterprise working directory, enter the following command in a command console:

   ```
   echo %MQSI_WORKPATH%
   ```
2. Edit the file to set the environment variable. for example, you can enter `MQSI_MAP_DB_PARAMETERMETADATA_SUPPORT = DatasourceName1:true DatasourceName2:false`.
3. Set the environment variable. For more information, see Setting up a command environment.

Calling a stored procedure from a map
Use the Graphical Data Mapping editor to call a stored procedure by using the Database Routine transform.

Before you begin

Before you start:

You must complete the following task:

- Create a graphical data map by using the Graphical Data Mapping editor. For information, see “Creating a message map” on page 1382.

About this task

You can use the Database Routine transform to call a stored procedure from a database.

Note: For information about the support for stored procedures, see “Support for stored procedures” on page 1318.

The Database Routine transform acts as a nested mapping, into which you can wire inputs to construct mappings to set the input parameter values for the stored procedure call. The routine input parameters

are displayed as outputs of the Database Routine nested mapping. You cannot wire any output from the Database Routine transform.

Use the Return transform within the Transform group of the Database Routine to map any output parameters, return values, or **ResultSets** produced by calling the Database Routine. You can wire any additional inputs that you want to map when the Database Routine call completes successfully. The output parameters, any return value, and any returned Result sets you defined for the Database Routine are provided as inputs in the nested Return mapping. You can wire the Return to a single or multiple sibling output elements to enable the returned data to be mapped to the map output.

Procedure

Using the Graphical Data Mapping editor, complete the following steps:

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click the canvas, and select **Database** > **Call Database Routine**.
   - Alternatively, select a schema element (or elements) as an input parameter value. Optionally, select a schema element (or elements) as an output value.
   - You can also use drag-and-drop to create the Database Routine transform. Connect the input object to the output object, and a transform is automatically created. Select the transform, and choose **Database Routine** from the **Transforms** list.
   - You can also click the **Call a stored procedure in a database** icon. 

   The Database Routine wizard is displayed.
2. In the **Database** field, select the database that you want to call the routine from. To add a database definition file, or to discover a new database by connecting the IBM App Connect Enterprise Toolkit using JDBC to a database server, click **Add database...**. For more information, see "Creating a database definition (.dbm file) by using the New Database Definition File wizard" on page 1475. You must select the **Routines** option during the discovery process.

   To use a different database name at run time than the name used in the IBM App Connect Enterprise Toolkit, you can override this value by setting the **Name of the database** property of the JDBC Providers policy that defines how to connect to your database; see JDBC Providers policy (JDBCProviders).
3. In the **Schema** field, select the database schema that defines the stored procedure that you want to call from the map.

   To call the stored procedure from a different database schema at run time, you can override this value by setting the **databaseSchemaNames** property of the JDBC Providers policy that defines how to connect to your database; see JDBC Providers policy (JDBCProviders).
4. In the **Routine** field, select the stored procedure that you want to call into the Database Routine transform in the map.
   a) If the selected routine can return a value, a **Return value** check box is displayed. If you want to make the return value available for mapping in the Return transform, check this box.

   Note: The selected **Routine** then populates the following locked fields:

   - **Type**: states whether the type of the selected **Routine** is a stored procedure .
   - **Parameters**: details of the parameter names, mode, and type for the selected **Routine**.
   - **Max ResultSets**: if the Database definition file for the **Routine** provides this information, states the maximum Result sets. Otherwise, it is blank.

   These fields can be displayed by clicking **Display Routine parameter details...**
5. Optional: Select **Treat warning as error**.

   If this option is selected, and calling the Database Routine in the configured runtime database returns an SQL warning, it is handled as if the database is raising an exception. If the Failure transform is

present, then it enters its nested mapping. Otherwise, the map execution stops, and an exception is raised from the Mapping node that is running the map.

Important: Database warnings are vendor-specific. For more information about database warnings, see the documentation for your database product.

6. Optional: If the selected **Routine** can return Result sets, and you want to map values from them, you must define their order and column contents.
   a) If your Database definition file defined the number of Result sets, then **Result sets list** is pre-populated with one **ResultSet**, and displays the maximum number that can be returned. Use the **Add** and **Delete** buttons to populate the **Result sets list**, up to any maximum number defined in the database definition file. You must order the Result sets as they are defined in the Database Routine code.
   If you only want to map data from, for example, the second result set, you must still include the first result set in the table because they are accessed by their positional order.
   b) Select each Result set, and use the check boxes in **Available table columns** to add column definitions to the Result set to match what the Database Routine returns. You only need to define the Result set columns that you want to be available for mapping.

   Note: If you want to map from a column in a specific table, and other tables have a column with the same name, you must add column definitions for all the columns that have the same name as the column that you want to be available for mapping. You must also add the columns in the same order that the columns appear in the result set.
7. Click **OK**.

   The Database Routine, and its grouped Return transform are displayed in your graphical data map. If you made any selections in the mapping input/output, all selected inputs are wired into the Database Routine, and outputs are wired to the Return transform. If you made no selections, then the new transform appears in the map unconnected.
8. Provide any required values for **IN** and **INOUT** mode parameters for the Database Routine.

   The input parameters for the selected stored procedure are displayed as outputs in the nested Database Routine transform.
   a) Connect the required input elements to the Database Routine, and within the nested map provide transforms to set a value for each parameter.

   The Database Routine is entered only once, making one call to the database system. You must set cardinality for any repeating elements that are connected into the Database Routine transform, or use a function transform to provide a single value to the parameter.

   Note: If any parameter is not given a value, the database server might return an exception if it cannot provide a default value. If the resulting output value of the transforms setting the parameters is not the correct type for the Database Routine, or the content is invalid, for example, exceeding a maximum length, a database exception might occur.
9. Provide any required mapping for the output elements from data that is returned from the Database Routine. This data can include **OUT** and **INOUT** mode parameters, optional **Routine** return values, and one or more Result sets. The **Routine** outputs are displayed as inputs in the nested Return transform.
   a) Optional: Connect any additional input elements that you might require merged with the Database Routine data to the Return transform. Connect the Return transform to one or more output elements of the map. Provide transforms within the nested Return map to set the connected outputs from the provided Database Routine output values.

What to do next

Next:

- If you want exceptions that are returned from the database server when the Database Routine is called to be handled by the map, instead of having such exceptions stop the map and being reported, you can add a Failure transform into the transform group; see "Handling database exceptions in a graphical data map" on page 1535.
- Set up a JDBC connection to the database that you want the run time to call the Database Routine into.

- If you want to modify the available columns in the Result sets for mapping, use the Properties view of the Database Routine transform, and then update the Return nested mapping.

Handling database exceptions in a graphical data map
Add a Failure transform to your graphical data map to handle exceptions that might be raised as a result of a database transform.

About this task

If you want the map to handle exceptions that are returned from the database server when the SQL operation is run, instead of such exceptions stopping the map and being reported, you can add a Failure transform to the transform group. The Failure transform is an optional transform in each of the database transform groups, and can be added or removed as required. If an exception is raised by the configured database server, and you have not configured a corresponding Failure transform, the map operation is stopped.

To add a Failure transform to a graphical data map by using the Graphical Data Mapping editor, complete the following steps:

Procedure

1. With a graphical data map (`.map`) file open in the Graphical Data Mapping editor, right-click a **Select**, **Insert**, **Update**, **Delete**, or **Database Routine** transform, and then select **Database** > **Handle Failure**.

   A **Failure** transform is created, and is displayed in your graphical data map.
2. Connect the **Failure** transform to specify how any exceptions from the database transform are processed when the map is run. If the Failure transform is present in the graphical data map, and is connected to one or more output objects, then the exception is caught and processed by the Failure transform. Database transforms additionally have a **Treat warnings as exceptions** option.
   - **Important:** If the Failure transform is present in the graphical data map, but is not connected, then the exception is caught by the Failure transform, and is ignored.
   - If the **Failure** transform has been deleted from the graphical data map, then the exception is handled by the Mapping node in your message flow, and is handled in the same way as other message flow exceptions.
3. Click the **Failure** transform to open the nested map and further define the transform.

Results

You have added and configured a **Failure** transform into your graphical data map. If you want the failure to cause the execution of the map to stop when the database transform receives an SQL exception, remove the **Failure** from the transform.

Behavior when modifying database column values from optional source elements
When updating or inserting database columns, you can define different behaviors for a missing, empty, or nil source.

Behavior with no Database Policy

When Inserting or Updating data into a database column by connecting a transform which is defined as optional in the schema model, you might want to consider the behavior for the possible source input states: Missing, Empty, or Nil. The behavior can be default or customized by enabling a Database Policy.

Table 1 defines the source states and the behavior without a Database Policy enabled.

Tables 2 and 3 define the behavior of enabling a Database Policy to check the source state and to then take a specific configured action.

Table 61. Behavior with no Database Policy on transforms linked to a column in an Insert or Update operation

Source state	Definition	Behavior
Missing Source	The input document does not contain the source element.	The column will not be passed in the SQL statement sent to the database server. The outcome is determined by the definition of the target column in the database: • If the column is defined with a default value, this value is set by the database system. • If the column is defined as nullable, and no default is defined, the column is set to null by the database system. • If the column is defined as not nullable, and no default is defined, the database will return a SQL exception.
Empty Source	The input document contains the source element, but that source is empty.	IBM App Connect Enterprise passes the value returned by "getValue" for the source element as the parameter value for the column in the SQL statement sent to the database. For example, an element of the String type will return the empty String value, so the target database column would be set with an empty string, "".
Nil Source	The input document contains the expected source, and it is nil.	The value returned by "getValue" is set to NULL.

Behavior for Insert with an enabled Database Policy

When Inserting data into a database column, you can enable a database policy on each transform mapping a single value from a source element. This allows you to choose one of the following actions for each of the input source states: Missing, Empty, or Nil.

Table 62. Behavior with a Database Policy enabled on transforms linked to a column in an Insert operation

Actions for source state	Behavior
Exclude column from database operation	Insert the database default value for the column. The column is excluded from the SQL statement sent to the database. This option is only enabled if the target database column has a default value defined in the database model from the associated `.dbm` file.
Insert the empty String value ""	This option is only enabled if the target database column is defined as any character string type in the database model from the associated `.dbm` file.
Set to NULL	This option is only enabled if the target database column is defined as nullable in the database model from the associated `.dbm` file.
Throw a map error	Produces a map error: • Missing: BIP3970 • Empty: BIP3971 • Nil: BIP3972

Behavior for Update with an enabled Database Policy

When Updating data in a database column, you can enable a database policy on each transform mapping a single value from a source element. This allows you to choose one of the following actions for each of the input source states: Missing, Empty, or Nil.

Table 63. Behavior with a Database Policy enabled on transforms linked to a column in an Update operation

Actions for source state	Behavior
Exclude column from database operation	The column is excluded from the SQL statement sent to the database. The value of the column currently in the database is not changed.
Set to the empty String value ""	This option is only enabled if the target database column is defined as a character string type in the database model from the associated `.dbm` file.
Set to NULL	This option is only enabled if the target database column is defined as nullable in the database model from the associated `.dbm` file.
Throw a map error	Produces a map error: • Missing: BIP3970 • Empty: BIP3971 • Nil: BIP3972

Referencing message maps in your solution

You can reference a message map and a submap during the development phase. You can also reference a message map dynamically at runtime.

About this task

Message maps and submaps are resources that can be used more than once in your solutions. For example, a message map can be used by one or more Mapping nodes in a message flow.

Procedure

Choose any of the following options to reference a message map or a submap in your solution:

- Reference a message map: You can reference a message map by configuring the **Mapping routine** property in a Mapping node. For more information, see "Referencing an existing message map from a Mapping node" on page 1537.
- Dynamically reference a message map at runtime: You can define a message map dynamically at runtime by setting the local environment **MappingRoutine** element in your message flow before the message reaches the Mapping node where the message map needs to be used. For more information, see "Dynamically selecting a message map" on page 1538.
- Reference a submap: You can call a submap from a graphical data map in the Graphical Data Mapping editor by using the **Submap** transform. For more information, see "Calling a submap" on page 1540.

Referencing an existing message map from a Mapping node

A message map can be used by one or more Mapping nodes in a message flow. You can reference a message map by configuring the **Mapping routine** in a Mapping node.

About this task

The Mapping node invokes a map-based transform.

The input to the Mapping node is the input message assembly that is propagated from the upstream node.

The output of the Mapping node is the new message assembly that is created by the mapping operations and propagated from the output terminal of the Mapping node.

Procedure

To reference an existing message map from a Mapping node, complete the following steps:

1. In the **Application Development** view, double-click the message flow that contains the Mapping node that you want to modify.

   The message flow opens in the Message Flow editor.
2. In the Message Flow editor, click the Mapping node that you want to modify.

   The Mapping node properties are displayed in the Properties view.
3. In the Properties view, select the **Basic** tab.
4. Next to the **Mapping routine** field, click **Browse**.

   The **Data Transformation Map Selection** window opens.
5. From the list in the **Data Transformation Map Selection** window, select the message map that you want to reference from your selected Mapping node, and click **OK**.

   Message maps are listed in the format `{`*BrokerSchemaName*`}:`*MapName*.

   `{default}` indicates that no broker schema is used by the message map.
6. Save and close your message flow.

Results

Your Mapping node references the message map that you selected.

Dynamically selecting a message map

To dynamically assign a message map to a Mapping node at runtime, you must pass the new map name in the local environment tree. You must define the new map name in the **MappingRoutine** element. The value you set in the **MappingRoutine** element overrides the map name that is set in the **Mapping routine** property of the Mapping node.

About this task

You can create, deploy, and run a message flow that invokes a different message map at a Mapping node.

You can override the mapping routine that is used to transform a message instance by specifying a new mapping routine in the local environment **MappingRoutine** element. You must specify the new mapping routine in the local environment tree that is upstream of the Mapping node that you need to modify.

The mapping routine qualified name that is provided in the **MappingRoutine** element must be defined in a map file that has to be deployed to the integration node in a BAR file where the message flow is deployed.

You can use any of the following message flow nodes to set the value of the local environment **MappingRoutine** element:

- Mapping node
- Compute node
- JavaCompute node

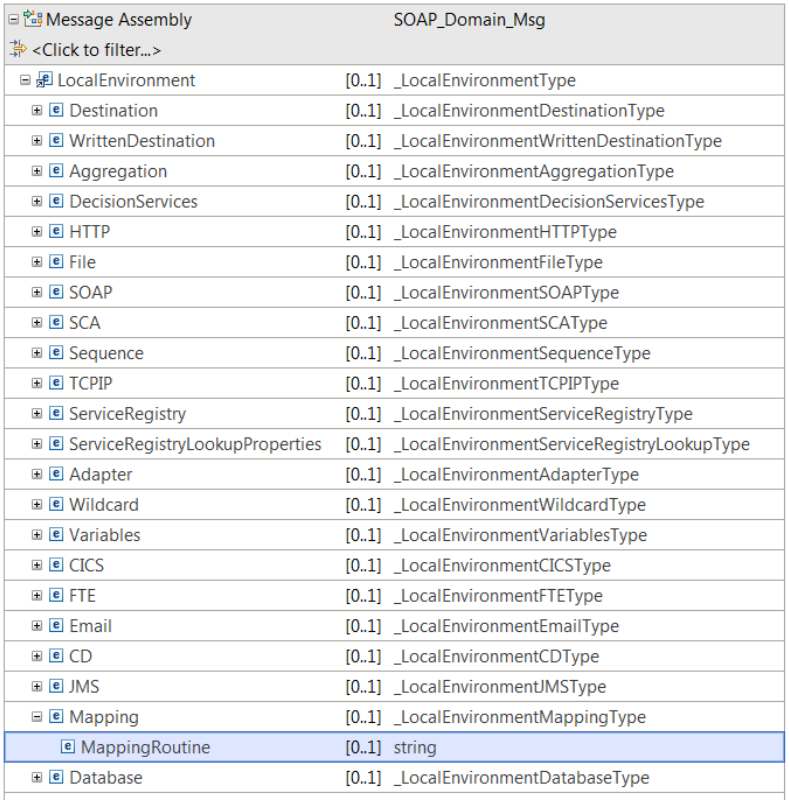
Procedure

To override at runtime the message map configured during development in a Mapping node, you must complete the following steps:

1. Required: Reference a message map in the Mapping node. This is the default message map executed by the Mapping node. For more information, see "Referencing an existing message map from a Mapping node" on page 1537.

   You must configure the name of the mapping routine that contains the statements to execute against the database or the message tree in the Mapping node property **Mapping Routine**. By default, the name that is assigned to the mapping routine is: `{default_broker_schema}:DefaultMsgFlowName_MappingNodeName`, where *default_broker_schema* is the broker schema where the message flow is located, and *DefaultMsgFlowName_MappingNodeName* is the name of the message flow concatenated with the name of the Mapping node.
2. In your message flow, add a new Mapping node located before the Mapping node where you want to assign dynamically a message map. Then complete the following configuration steps in the new Mapping node:
   a) Add the local environment tree to the input message assembly.
   b) Add the local environment tree to the output message assembly.
   c) Optional: Add database tables if you require information available in an external database.
   d) Configure a **Move** transform between the input local environment tree and the output local environment tree.
   e) Configure a transform to set the output local environment tree **MappingRoutine** element. You can also add a condition to the transform.

      You must define the value for **LocalEnvironment** > **Mapping** > **MappingRoutine**.



      A light bulb will appear on the left hand side of the transform.
   f) Click the yellow light bulb, and then select **Group the conflicting transforms in an override group**.

Results

You have configured your message flow to dynamically assign a message map to a Mapping node. In the first Mapping node, you have defined the logic to set the local environment **MappingRoutine** value. In the Mapping node where you want to dynamically assign a message map, you have defined a message map.

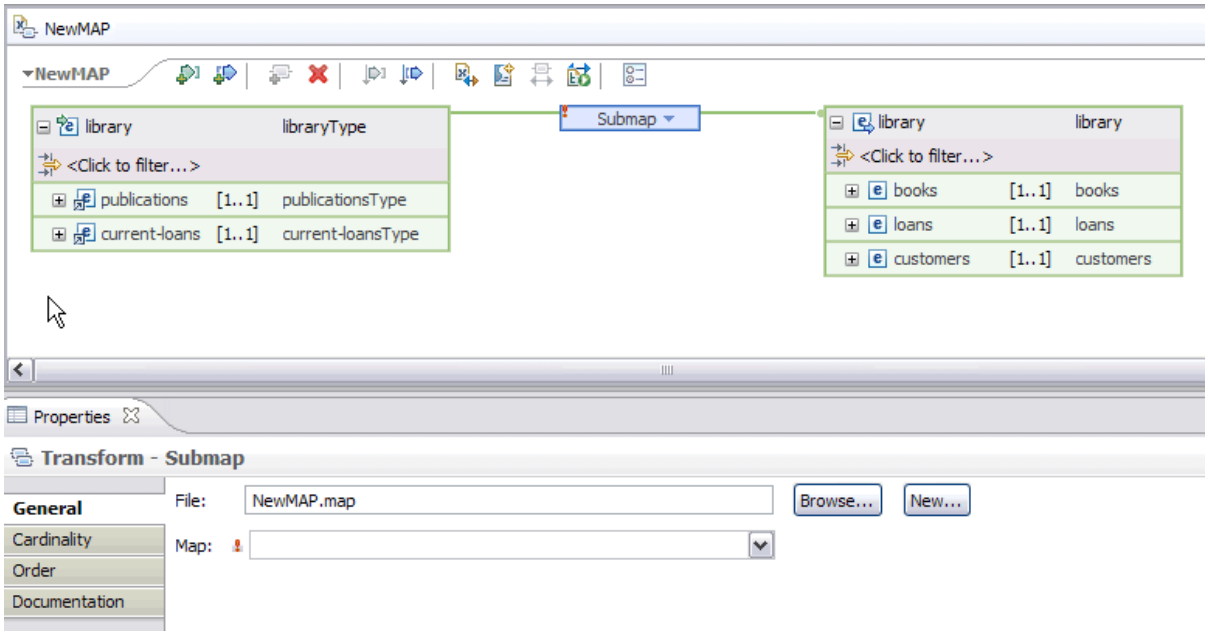
Calling a submap
You can call a submap from a graphical data map in the Graphical Data Mapping editor by using the **Submap** transform.

Procedure

Complete the following steps to call a submap from another graphical data map:

1. Create a connection between global input and output elements in a graphical data map, and then select the **Submap** transform on the connection:

   For example:



2. Click **Browse** in the Properties view of the **Submap** transform.

   The **Submap Selection** wizard is displayed where the dialogue box will display the submaps that are available.

3. Select a submap, and click **OK**.

   You can choose to display only the valid maps that can be selected, by clicking **Show only applicable maps**.

Results

The submap is displayed in the Graphical Data Mapping editor, and you can edit it in the same way that you would edit any graphical data map. For information about how to edit maps, see "Editing message maps" on page 1394.

Transforming a SOAP message in a message map

In IBM App Connect Enterprise, a SOAP message is described by a generic model that includes the SOAP *Envelope* and optionally *Attachments*. You define your SOAP message parts in a message map by using the **Cast** function.

About this task

A SOAP message consists of an *Envelope* and optionally *Attachments*. The envelope contains a SOAP header and a SOAP body. A SOAP body can include SOAP faults.

When you use SOAP nodes in IBM App Connect Enterprise, you can manipulate the following type of messages:

- A SOAP message that is described by a generic model and that contains the SOAP operation-specific data.
- An XML message that contains only the SOAP operation-specific data.

In addition to the standard SOAP parts, the SOAP message generic model includes a *Context* part that includes contextual information about the current SOAP message that is processed. This part is the only one in a message map whose structure is included automatically. You must define the other SOAP message parts manually by using the **Cast** function.

IBM App Connect Enterprise supports SOAP 1.1 and SOAP 1.2 messages.

Depending on the message domain that you configure in your input node, you might have to consider the differences between SOAP 1.1 and SOAP 1.2 when transforming SOAP messages.

- If you receive a SOAP message through a SOAPInput node, the SOAP parser handles SOAP 1.1 or SOAP 1.2 automatically. The SOAP domain uses a common logical tree format that is independent of the exact format of the web service message. For details of the SOAP tree format, see SOAP tree overview.
- If you receive a SOAP message through an HTTPInput node, the XMLNSC parser handles your SOAP 1.1 or SOAP 1.2 message differently. When you create a message map, you must be aware of the SOAP version, and configure the correct SOAP 1.1 or SOAP 1.2 schema when you create and configure your graphical data map.

Depending on the nodes that you use when you model your message flow or your service operation, and the message domain you configure, you must use a different schema model:

- If you use the SOAP nodes excluding the SOAPExtract node, you must map the **SOAP_Domain_Msg** in the SOAP domain.
- If you use the SOAP nodes including the SOAPExtract node, and the Mapping node is wired after a SOAPExtract node, you must map the schema associated with your operation in the XMLNSC domain. You use the SOAPExtract node to remove SOAP envelopes, allowing just the body of a SOAP message to be processed.
- If you use HTTP nodes or MQ nodes, you must map the SOAP 1.1 or the SOAP 1.2 schema as the root model of the map in the XMLNSC domain.

The following table summarizes the different types of nodes and domains that you can use to map a SOAP message and the schema that you must use when you use a message map to transform a SOAP message.

Table 64. Schemas to use when transforming a SOAP message

Message domain	Nodes usage	Schema to configure in a message map
XMLNSC	All SOAP nodes within an integration service operation subflow.	Schema for operation request or response.
XMLNSC	SOAPInput nodes in flows that have a SOAPExtract node before the Mapping node.	Schema for operation request.

Table 64. Schemas to use when transforming a SOAP message (continued)		
Message domain	**Nodes usage**	**Schema to configure in a message map**
XMLNSC	SOAPReply or SOAPRequest nodes that might be preceded in a flow by a SOAPEnvelope node.	Schema for operation request or response.
SOAP	SOAPInput nodes that are wired directly to the Mapping node.	IBM-provided SOAP_Env where the body `xsd:any` has a mapping cast to the schema for the operation request.
SOAP	SOAPReply or SOAPRequest nodes that are wired directly to the Mapping node.	IBM-provided SOAP_Env where the body `xsd:any` has a mapping cast to the schema for the operation request or response.
XMLNSC	Flows that do not contain SOAP nodes. For example, flows that contain HTTP nodes, MQ nodes, or FileInput nodes.	IBM-provided SOAP 1.1 or SOAP 1.2 schema where the body `xsd:any` has a mapping cast to the schema for the operation request or response.

The following table compares the SOAP message structure with the IBM App Connect Enterprise SOAP message generic model:

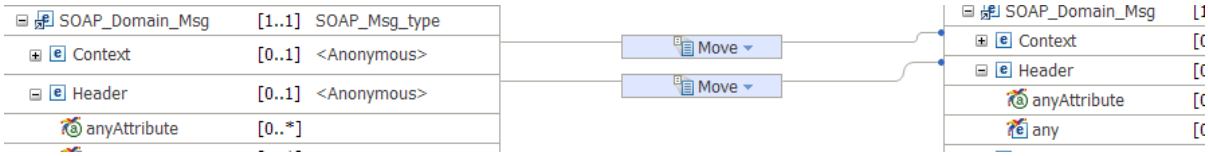
Table 65. Comparison between the SOAP message structure and the IBM App Connect Enterprise SOAP message representation			
Standard SOAP message parts	**Status**	**IBM App Connect Enterprise SOAP message parts**	**IBM App Connect Enterprise Status**
		Context	Required
SOAP header (part of the SOAP envelope)	Optional	Header (part of the SOAP_Domain_Msg)	Optional
SOAP body (part of the SOAP envelope)	Required	Body (part of the SOAP_Domain_Msg)	Required
SOAP faults (part of the SOAP body)	Optional	Fault (part of the Body)	Optional
SOAP Attachments	Optional	Attachment (part of the SOAP_Domain_Msg)	Optional

Procedure

Complete the following steps to configure the **SOAP_Domain_Msg** when the Mapping node is connected directly from a SOAPInput node with no SOAPExtract node:

1. Define the transformation for the **Context** object by using one of the following approaches:
   - Copy the **Context** object by using the **Move** transform.
   - Copy the **Context** object by using the **Move** transform, and then use the **Override** function if you want to modify a few elements with an **Assign** transform or a **Move** transform.
   - Define transforms for all the **Context** elements that you want to maintain in the output object.

2. Define the transformation for the **Header** object by using one of the following approaches:
   - Copy the **Header** object by using the **Move** transform.



   - Copy the **Header** object by using the **Move** transform, and then use the **Override** function if you want to modify a few elements with an **Assign** transform or a **Move** transform.
   - Define transforms for all the **Header** elements that you want to maintain in the output object.

   You define SOAP header parts by using the **Cast** function. You can cast attributes and other header parts. Then, define transforms between the input elements and the output elements in each header part.

   The SOAP Header element contains application-specific information, including attributes that define how you process the SOAP message.
3. Define the transformation for the **Body** object.

   You define SOAP body parts by using the **Cast** function. You can cast attributes and other body parts. Then, define transforms between the input elements and the output elements in each body part.

   Complete the following steps to define the SOAP body parts and their transformations:

   a) Cast attributes that are defined as *xsd:any* into a specific type. Then, define a transform between the input attribute and the output attribute.

   b) Cast wildcard elements that are defined as *xsd:any* into a specific type.

   c) Cast a base type element to a derive type element. A derive type element is also known as an extension type element.

      In a message map, you cast a base type to a derive type or extension type so that you can define transformations between subtypes of a data type.

      For example, addresses are represented differently for different countries. You might want to map addresses from different countries into a common complex structure for addresses.

   d) Define transforms between the input and output body elements.

   Note: When you use a schema model for your SOAP message or the built-in SOAP domain message, you can also define the **xsd:any** by using the **Add User-Defined** function. However, you are recommended to use the schema models that are part of the WSDL service definition, if they are available.
4. Define the transformation for the **Attachment** object by using one of the following approaches:
   - Copy the **Attachment** object by using the **Move** transform.
   - Copy the **Attachment** object by using the **Move** transform, and then use the **Override** function if you want to modify a few elements with an **Assign** transform or a **Move** transform.

   You define SOAP attachment parts by using the **Cast** function.

Example

Complete the steps in any of the following use cases to learn how to configure the **SOAP_Domain_Msg** when the Mapping node is wired directly from a SOAPInput node with no SOAPExtract node:

- Configure a SOAP message when you use a conditional transform to map an input element to an output element. For example, you create and configure the **If, Else if, and Else** transform to control the flow of the mapping between elements that are defined as a specific or a derive type in the input and output message assembly by setting conditions. For more information, see "Mapping a SOAP message by using a conditional transform" on page 1544.

- Configure a SOAP message to transform some input elements to output elements. Use the **Override** function, **Assign** transform, and **Move** transform. For more information, see "Mapping a SOAP message by using the Override function" on page 1546.

What to do next

Define more transforms between the input **SOAP_Domain_Msg** and the output **SOAP_Domain_Msg**. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Mapping a SOAP message by using a conditional transform
You define SOAP message parts in a message map by using the **Cast** function and then you define transforms between its elements. To map between elements that are defined as specific or derived types, you might want to define some conditional transforms. You can configure the **If, Else if, and Else** transform to control the flow of the mapping between elements.

About this task

When you use an **If, Else if, and Else** transform between your **SOAP_Domain_Msg** input object and **SOAP_Domain_Msg** output object, you must manually configure each element in the **SOAP_Domain_Msg**. You must map each element in the **SOAP_Domain_Msg** input object to the corresponding output object so that you do not lose the information of the element.

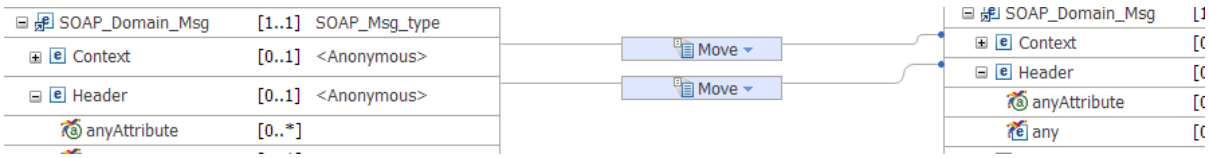
Note: Elements that are part of the input object and do not have a transform that is defined to an output object are deleted from the output structure and their value is lost.

Procedure

Complete the following steps to configure the **SOAP_Domain_Msg** when the Mapping node is wired directly from a SOAPInput node with no SOAPExtract node:

1. Define the transformation for the **Context** object.

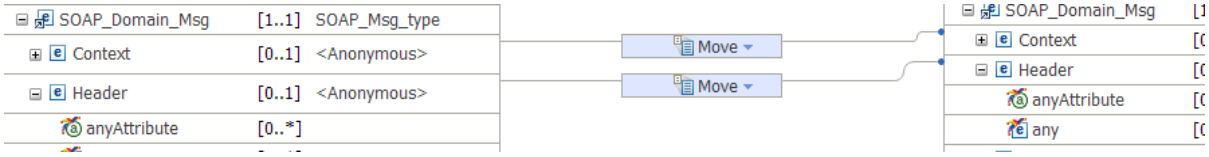
   Copy the **Context** object by using the **Move** transform. For more information, see "Specifying a transform (mapping operation)" on page 1430.



2. Define the transformation for the **Header** object.

   You can define a **Move** transform between the input **Header** object and the output **Header** object to copy the structure and all its elements.

   Copy the **Header** object by using the **Move** transform.



3. Define the transformation for the **Body** object. Define SOAP Body parts by using the **Cast** function. You can cast attributes and other body parts. Then, define transforms between the input elements and the

output elements in each body part. For more information, see "Casting elements in a message map" on page 1398.

a) Cast attributes that are defined as *xsd:any* into a specific type. Then, define a transform between the input attribute and the output attribute.

b) Cast wildcard elements that are defined as *xsd:any* into a specific type.

c) Cast a base type element to a derive type element. A derive type element is also known as an extension type element.

In a message map, you cast a base type to a derive type or extension type so that you can define transformations between subtypes of a data type.

For example, addresses are represented differently for different countries. You might want to map addresses from different countries into a common complex structure for addresses.

d) Create and configure the **If, Else if, and Else** transform to control the flow of the mapping between elements that are defined as a specific or a derive type in the input and output message assembly by setting conditions.

4. Optional: Define the transformation for the **Attachment** object. Copy the **Attachment** object by using the **Move** transform.

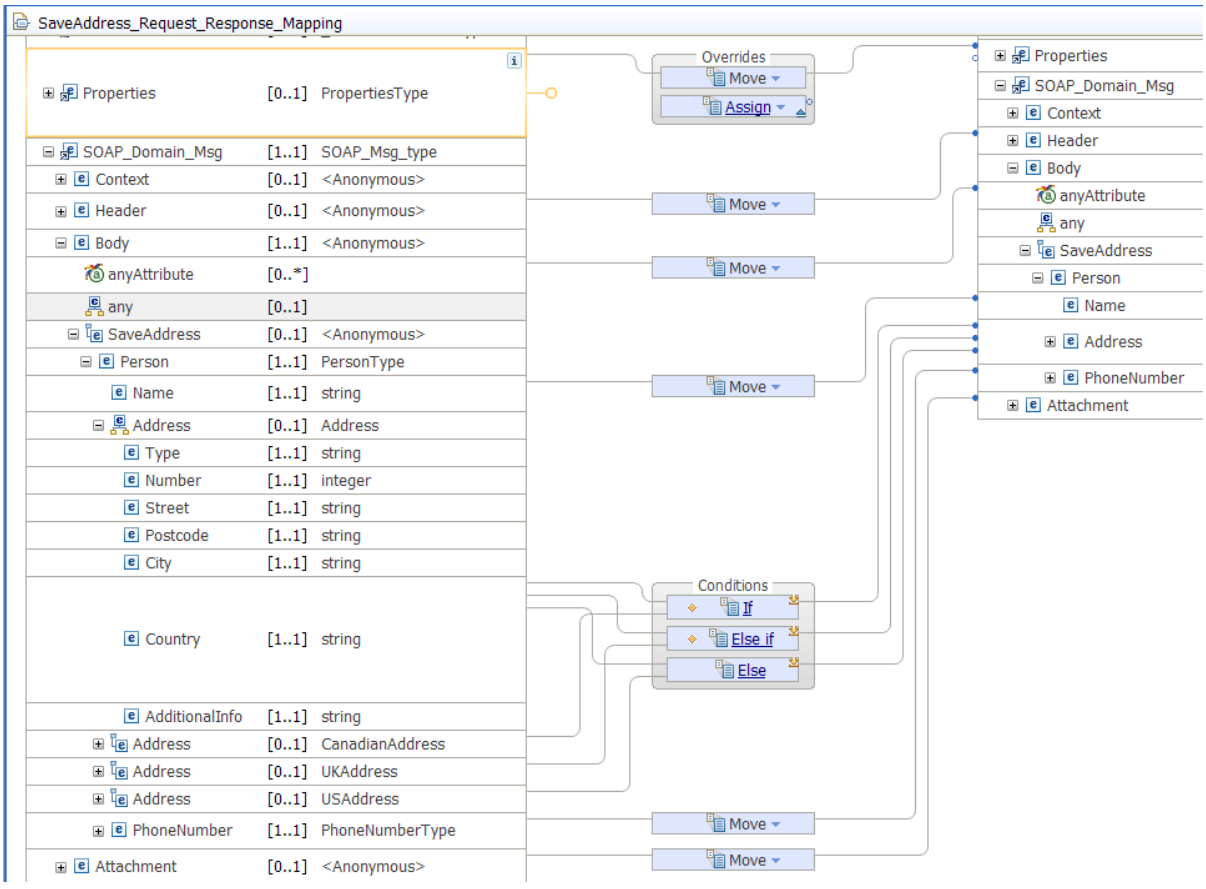
You define SOAP attachment parts by using the **Cast** function.

Results

You have a message map that transforms a SOAP message. The message map contains a nested map that uses the **If, Else if, and Else** transform.

Example

The following figure shows a message map after you complete the previous steps to transform a SOAP message:



What to do next

Define more transforms between the input **SOAP_Domain_Msg** and the output **SOAP_Domain_Msg**. For more information, see "Specifying a transform (mapping operation)" on page 1430.

*Mapping a SOAP message by using the **Override** function*
In a SOAP message, you can use the **Override** function to copy a complex type from the input message to the output message, while you update some of the child elements in the complex type with an **Assign** transform, and a **Move** transform.

Before you begin

You define your SOAP message parts in a message map by using the **Cast** function. For more information, see "Casting elements in a message map" on page 1398.

About this task

Note: You can include **Move** transforms and **Assign** transforms within an **Override**.

Procedure

Complete the following steps to configure the **SOAP_Domain_Msg** when the Mapping node is wired directly from a SOAPInput node with no SOAPExtract node:

1. Define a **Move** transform to copy all the elements in the input **SOAP_Domain_Msg** message unchanged.
2. Change the value of at least one element by using an **Assign** transform or a **Move** transform.
3. Apply a quick fix to configure the **Override** function. Select the option **Group the conflicting transforms in an override group**.

Results

You transformed the SOAP message by using the **Override** function.

What to do next

Define more transforms between the input **SOAP_Domain_Msg** and the output **SOAP_Domain_Msg**. For more information, see "Specifying a transform (mapping operation)" on page 1430.

Creating or transforming a BLOB output message by using a graphical data map

Use the Graphical Data Mapping editor to create or transform a message using the BLOB message domain.

About this task

When you create a new map, the BLOB message is available under the "IBM supplied message models" category in the map input and output selection tree. If you select a BLOB message as the map output, the output domain in the output message assembly is automatically set to BLOB. For more information about creating a new map, see "Creating a message map from a Mapping node" on page 1385.

To populate the data of a BLOB message, you must wire a transform that will provide hexBinary formatted data to the child 'value' element of the message. You can use the built-in function `iib:hexBinaryValue($<var>)` in a Custom XPath transform, to format data into the required hex binary string.

When you transform a BLOB message, you must wire the value element of the input BLOB message into a transform that accepts an `xsd:hexBinary` type. For example, you could use a Move transform to set the BLOB binary data into an XML output message element that has either an `xsd:hexBinary` or `xsd:base64Binary` type.

If you are transforming only the Message Assembly headers and folders (for example, Properties and Transport headers), you can pass the message body data through unmodified by using the BLOB domain and a Move transform from the BLOB input message to the BLOB output message.

Mapping from a BLOB message to an output message using a graphical data map

You can use the Graphical Data Mapping editor to transform a message in the BLOB message domain.

About this task

When you create a new map, the BLOB message is available under the "IBM supplied message models" category in the map input and output selection tree. Select the BLOB message as the map input, and then select your required output message and set its domain in the output message assembly (for example, `XMLNSC`).

For more information about creating a new map, see "Creating a message map from a Mapping node" on page 1385.

The BLOB message provides a single element value of type `xsd:hexBinary`. When you transform a BLOB message, you must wire the value element of the input BLOB message into a transform that accepts an `xsd:hexBinary` type. For example, you could use a Move transform to set the BLOB binary data into an XML output message element with either an `xsd:hexBinary` or `xsd:base64Binary` type.

Creating or transforming a JSON message by using a message map

You can use the Graphical Data Mapping editor to create or transform a JSON object message or a JSON array message. You can provide the model of your JSON message in a JSON schema file or a REST API swagger document or an OpenAPI 3.0 document. Alternatively, you can define your JSON data by using the **Add User Defined** function in the map.

About this task

When you create a message map, you can choose to provide the model of the JSON data from a JSON schema or REST API Swagger document or an OpenAPI 3.0 document in a supported project container. Alternatively, you can define your JSON data by using the **Add User Defined** function in the map, or an equivalent XML schema model.

When you work with a JSON model, you can use a JSON schema, a Swagger document, or an OpenAPI 3.0 document in either JSON or YAML format. You can also use the JSON elements and types to perform mapping casts. For more information, see "Casting elements in a message map" on page 1398.

Procedure

Follow the steps in either of the following topics to create or transform a JSON message:

- "Creating or transforming a JSON message by using a JSON schema" on page 1547
- "Implementing a REST API operation by using a message map" on page 1580
- "Creating or transforming a JSON output message by using the User Defined function to model the JSON data" on page 1552
- "Modeling a JSON message for use in a message map by using an equivalent XML schema model" on page 1572

Creating or transforming a JSON message by using a JSON schema

You can use the Graphical Data Mapping editor to create or transform a JSON message with a data model defined from a JSON schema or a Swagger document.

Before you begin

Ensure that you have a JSON schema or Swagger document to provide the model for your data. If you do not have a JSON schema model, you can model and transform your data using the **Add User Defined** function, as described in the following topics:

- "Graphically modeling a JSON object in a message map" on page 1557
- "Graphically modeling a JSON array in a message map" on page 1560

About this task

When you create a message map for a Mapping node, you can choose to have the input and output message body data defined through a JSON type in a supported JSON schema.

When you select this option, the map is created with the input and output data fully defined by the JSON schema data definitions in the referenced JSON schema file. When you create a message map to transform JSON data in this way, the map must be located in the same shared library or REST API as the JSON schema file.

When working with a JSON model, you can use a JSON schema or Swagger document in either JSON or YAML format. You can also use the JSON elements and types to perform mapping casts; for more information, see "Casting elements in a message map" on page 1398.

Procedure

Use a message map to transform data to or from a JSON data model in a JSON schema, by following these steps:

1. Ensure that your JSON schema model is available for use by the message map:
   a) Put your JSON schema file (with a `.json`, `.yaml`, or `.yml` extension) in one of the following containers:
      - Application
      - REST API project
      - Integration Service project
      - Shared library
      - Static library

      Note: Your message map and JSON schema file must be stored in the same project.

      Ensure that the schema model file conforms to the JSON Schema draft 4 specifications and meets the requirements specified in "JSON schema requirements for message maps" on page 1549.
   b) Ensure that the container selected to create the new message map is the one that contains the JSON schema or Swagger documents that need to be referenced. Also ensure that the model definitions meet the requirements specified in "JSON schema requirements for message maps" on page 1549.
2. Create a new message map by starting the **New Message Map** wizard, either from the context menu or by double-clicking a Mapping node. If you are creating a message map for a REST API operation request or response mapping, ensure that you start the **New Message Map** wizard by double-clicking

the Mapping node in the operation subflow. For more information, see "Implementing a REST API operation by using a message map" on page 1580.

a) On the first panel of the wizard, select a map that is called from a Mapping node; submaps are not supported with models from JSON schema.

b) Ensure that the container selected for the new message map is the same shared library or REST API project that contains your JSON schema file.

c) Click **Next** to move to the **Select inputs and outputs** panel. Expand the **JSON Types** folder under the shared library or REST API project folder, to view and select the required type. You can also enter part of the Type name in the filter boxes, to limit the search.

- JSON array types are prefixed with `JSONArray_`.
- The root level type from a JSON schema file is the basename of the containing file, suffixed with `_JsonType`.
- The type names are unique in a JSON schema file. If two JSON schema files provide a type with the same name, you must select the required type by using the filename to distinguish them.
- If your JSON data is not located in the message body (for example, if you have a REST request node that is configured to place its response in the environment), you can add a mapping cast to specify `Environment.Variables.any` when the map is created and opened in the editor.

d) If you have selected a JSON type that is not in the same Toolkit container as the one in which the map will be created, an error is displayed and you cannot continue until all the map and JSON schema files are located in a single shared library or REST API container.

When performing a mapping cast, the elements and types from JSON schema and Swagger documents located in the same container as the map will be included in the list of available options.

e) Click **Finish**, and the map is opened in the Graphical Data Mapping editor.

3. Complete the required transformation logic in the message map, by using any of the available transform types. For more information, see "Transform types in the Graphical Data Mapping editor" on page 1282 and "Editing message maps" on page 1394.

   In addition to transforming the message data, you can interact with resources as described in the following topics:

   - "Mapping database content" on page 1523
   - "Using Java in a message map" on page 1370
   - "Accessing user-defined properties from a Mapping node" on page 1479

What to do next

Deploy the solution with the message map, and exercise it to verify that the output message is as required. For more information, see "Troubleshooting a message map" on page 1587.

JSON schema requirements for message maps

You can use the Graphical Data Mapping editor to create and transform JSON messages with the data model that is defined from a JSON schema or REST API swagger document, which can be in JSON or YAML format.

The following requirements apply to both JSON schema and to the model definitions from an OpenAPI v3 or Swagger v2 document. For more information, see Swagger.

- JSON schema must be well formed, valid, and must conform to the JSON Schema draft 4 specifications.

  You can use an external tool to validate the JSON schema file before you import it and use it for message maps. For example, see http://jsonlint.com.

- JSON schema must be in either JSON format with a `.json` file extension, or YAML format with a `.yaml` or `.yml` file extension.

- JSON schema must be located in one of the following project containers:
  - Application
  - REST API project
  - Integration Service project
  - Shared library
  - Static library

  Note: Your message map and JSON schema file must be stored in the same project.
- When you are using a JSON schema for a Mapping Cast, such as mapping the output of a REST request node that is configured to place its output in environment tree, bear in mind that cast types and elements are provided for only the following types:
  - Root-level JSON type, with the same name as the schema file
  - Top-level JSON types
  - JSON types defined in the `components.schemas` section of an OpenAPI v3, or the `definitions` section of a Swagger 2 document.

  The elements for a JSON array are suffixed with `JSONArray`, and the complex types for a JSON array are prefixed with `JSONArray`.
- Heterogeneous JSON arrays, in which the type of each entry can vary, appear in the message map with the `Item` field that represents an array entry that is defined as an `any` type. You can cast this `any` type to the heterogeneous JSON array types only if the JSON schema defines them through JSON references.
- While external "$ref"s are supported in OpenAPI v3 documents, the JSON schema `"$ref"` keyword is supported only for internal references in a Swagger 2 document.
- The JSON schema "not" keyword, which can be used to exclude specific types from a JSON object, is not supported.
- To enable message mapping, all JSON data types that are to be mapped from or to must be explicitly modelled.
  - You cannot map from or to JSON data if it is present only because the JSON schema keyword `additionalProperties` is set to `true`. To be able to graphically map to or from data, it must be explicitly present in the model so that it appears as a named input or output in the map. Therefore, it is beneficial to use JSON schema models that set: `"additionalProperties": false`.
  - A JSON empty object, *"{}"* appears in the map as an element "_matchAnything", and allows a mapping cast to be applied. For more information, see Casting with JSON schema types.
- It is good practice to enable JSON domain message validation when you develop and test maps.
- For example:
  - A JSON schema data member with type `"string"` and property `"maxLength": 10`

    This type can have a transform that is applied in the map that sets the length in excess of 10 characters. This would not cause a map design validation or runtime error.
  - A JSON enum can be set incorrectly to a value not matching one of the enumerations.
- JSON schema data defined as type `"number"` is processed in the Mapping node by using a Message Assembly tree type of `"double"`. This might lead to the output JSON text providing the data in a scientific (exponential) notation. To avoid this, use the JSON schema type `"integer"`.
- In the message map, optional/required/number of occurrences is modelled as `"minOccurs"` and `"maxOccurs"`. The JSON schema keyword `"required"` sets `"minOccurs=1"`, otherwise it is set as `"minOccurs=0"`
- A JSON schema `"enum"` with mixed type entries appears as a `"string"` type in the message map.

- The JSON schema "oneOf" and "anyOf" keywords, which are used for structuring and combining more complex JSON schema, are supported as follows:
  - Specifying a JSON type or null:
    - To define a type as either a primitive or null. For example:

```
"NullableNums": {
       "oneOf": [
         {
           "type": "integer",
         },
         {
           "type": "null"
         }
       ]
}
```

    - To define a type as either an object or null, where the object can be defined inline or through $ref. For example:

```
"Address" : {
 "oneOf": [
  {"type": "null"},
   {"$ref":"#/definitions/Address"}
         ]
}
```

    In these cases, the map presents the JSON type in the map ready for wiring transforms. The properties show that the type is defined as nillable.
  - To define a JSON schema top-level type as an object or array of a $ref type. For example:

```
  "oneOf": [
    {
      "$ref": "#/definitions/Account"
    },
    {
      "type": "array",
      "items": {
        "$ref": "#/definitions/Account"
      }
    }
  ]
```

    In this case the New Message Map wizard, or the Map Editor Add Input or Output dialog provides two entries for the JSON type, one as an Object and the other as an Array (in the example, these are Account and JSONArray_Account).
  - Support for the "oneOf" and "anyOf" keywords was enhanced to handle a list with multiple types. For example:

```
"Contact" : {
"properties": {
"conType": {
"type": "string"
}
},
"oneOf": [
            {"type": "null"},
            {"$ref":"#/definitions/Address"},
            {"$ref":"#/definitions/TelContacts"}
        ]
        }
```

    In this case the map initially displays the JSON object "Contact" with only the "conType" child, or with no content when the JSON object has no properties. You can right-click "Contact" and apply a mapping cast to one or more of the types from the list that is provided in the JSON schema; in this example, the types are "Contact_Address" and "Contact_TelContacts". For more information, see “Casting with JSON schema types” on page 1405.

- The JSON data element names must also be a valid XML name.

  Note:

  Naming issues might not be detected in the Integration Toolkit, but they might lead to the deployment of the message map to fail with a `Map script generation` error with a cause of `invalid grammar`.

If these conditions are not met, you might see the following effects:

- When you create a new message map, the JSON types from the JSON file are not offered in the **New Map Wizard**.
- When creating a message map for a REST API operation subflow, the option **Message map with input and output for REST API operation** is offered, but when clicking **Finish** the wizard does not complete and an error is displayed.
- If a JSON schema file that is used by a message map is modified such that it is no longer supported, the message map is marked with an error that reports the JSON type from the JSON schema file is now unresolved. For example:

  ```
  Unable to locate object named "mlib://SafesForceAllModelsShlib/Contact#/-/simpleType{http://
  www.ibm.com/iib/msl/json}Contact". LeadToContact.map Map Reference Problem

  The null input or output object /Contact.json could not be found. LeadToContact.map Map
   Problem
  ```

  When the JSON schema file that is used by a message map is modified and the map has not been edited and saved again, you might have to invoke the `clean` action through the menu option **Project** > **Clean**.
- The cause of the failure to process a JSON schema for a message map is logged in the Integration Toolkit Eclipse Error Log.

Creating or transforming a JSON output message by using the User Defined function to model the JSON data
You can use the **Add User Defined** function in the Graphical Data Mapping editor to create or transform a JSON object message or a JSON array message.

About this task

When you create a message map, you can choose to provide the model of the JSON data from a JSON schema, or you can define your JSON data by using the **Add User Defined** function in the map. This topic explains how to use the **Add User Defined** function. For information about using a JSON schema, see “Creating or transforming a JSON message by using a JSON schema” on page 1547.

Predefined models are available under the IBM supplied message models category in the map input and output selection tree.

- You select **JSON (JSON object message model supplied by IBM)** to create a JSON object message.
- You select **JSON (JSON array message model supplied by IBM)** to create a JSON array message.

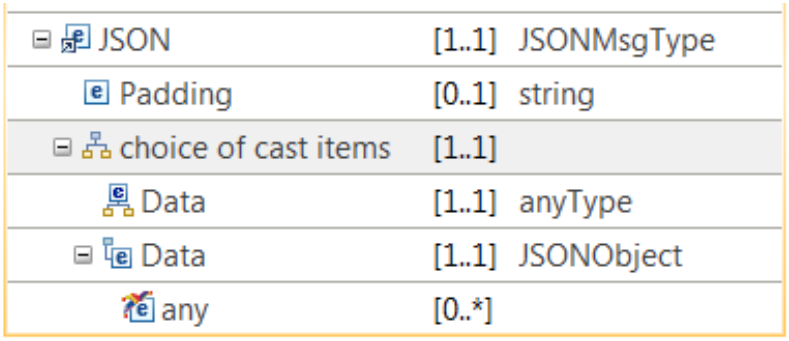
When you select any of the JSON message models as the map output, the output domain in the output message assembly is automatically set to JSON.

The predefined JSON message contains a single element value of type **JSONMsg_type**. This element contains the following child elements:

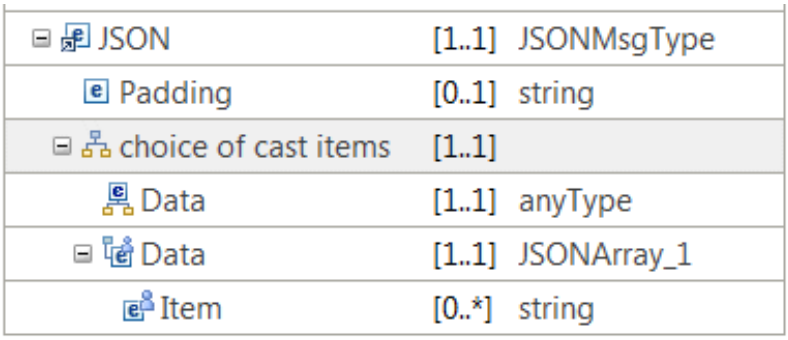
1. A **Padding** element: This element is optional. You define its value to pass a user-defined JavaScript function call used by JSONP services.
2. Two **Data** elements: You define only one of the **Data** elements to model your JSON message.
   - You define the element **Data**, of type **anyType**, by using the **Cast** function. You define this element when you have a schema model that defines your JSON message. If you create your map

programmatically, you can cast **Data** to some type in an external schema file, which can define either a JSON object or a JSON array message.

- You define the element **Data**, of type **JSONObject**, by using the **Add User-Defined** function. You define this element when you want to transform some elements of your JSON object message. You can also use the **Cast** function to define the JSON message, or a part of it, with a schema model.



- You define the element **Data**, of type **JSONArray_1**, by using the **Add User-Defined** function. You define this element when you want to transform some elements of your JSON object message. You can also use the **Cast** function to define the JSON message, or a part of it, with a schema model.



Procedure

You can complete any of the following actions on JSON messages in the Graphical Data Mapping editor:

1. You can define a JSON object without a schema model graphically. Use the **Add User-Defined** function to add an element, and select one of the supported simple types. For more information, see "Graphically modeling a JSON object in a message map" on page 1557.
2. You can define a JSON array without a schema model graphically. All the elements in the array are of the same type. Use the **Add User-Defined** function to add an element, and select **JSONArray_Default** as its type. For more information, see "Graphically modeling a JSON array in a message map" on page 1560.
3. You can define a multidimensional JSON array without a schema model graphically. Use the **Add User-Defined** function to add multiple elements, and select **JSONArray_Default** as their types. For more information, see "Graphically modeling a multidimensional JSON array in a message map" on page 1565.
4. You can create a JSONP message, or transform a message that is modeled in a different message domain into a JSONP message. For more information, see "Graphically modeling and transforming a JSONP message in a message map" on page 1579.
5. You can define a JSON message by using your own schema model. Use the **Cast** function to add a JSON message. For more information, see "Modeling a JSON message for use in a message map by using an equivalent XML schema model" on page 1572.
6. You can use the Graphical Data Mapping editor to implement a REST API operation. For more information, see "Implementing a REST API operation by using a message map" on page 1580.

Graphically modeling a JSON message in a message map
In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to create a JSON message that contains JSON objects, JSON arrays, or both.

About this task

If you have a JSON schema model, you can use it in your map in a shared library or REST API; for more information, see "Creating or transforming a JSON message by using a JSON schema" on page 1547. If you do not have a JSON schema model, you can create a map selecting the predefined JSON message and define your JSON data by using the **Add User-Defined** function as described in this topic.

Note: You cannot use both JSON schema and user-defined elements a message body. However, you can create a map that has the input message body defined by JSON schema, and the output message or local environment defined by using the **Add User-Defined** function.

Procedure

Complete the following steps to create a JSON message for which you do not have a schema model:

1. Create a message map. Select one of the following JSON supplied models as the input or output of the map:
   - **JSON (JSON object message model supplied by IBM)** to create a JSON object message
   - **JSON (JSON array message model supplied by IBM)** to create a JSON array message

   For more information, see "Creating a message map" on page 1382.
2. Qualify the content under the **Data** element by using the **Add User-Defined** function.

   A JSON message consists of name-value pairs, which are known as JSON objects, and ordered collections of values, which are known as JSON arrays.

   a. To define a JSON object in the JSON output message, use the **Add User-Defined** function to add an element, and select one of the supported simple types. For more information, see "Graphically modeling a JSON object in a message map" on page 1557.

   b. To define a JSON array in the JSON output message, use the **Add User-Defined** function to add an element, and select **JSONArray_Default** as its type. For more information, see "Graphically modeling a JSON array in a message map" on page 1560.

   c. To define a multidimensional JSON array in the JSON output message, use the **Add User-Defined** function to add multiple elements, and select **JSONArray_Default** as their types. For more information, see "Graphically modeling a multidimensional JSON array in a message map" on page 1565.

   Note: When you use the Graphical Data Mapping editor to transform a JSON message, ensure that the JSON element names that you define by using the **Add User-Defined** function are valid according to the XML rules for naming elements. JSON allows a wider range of characters than XML, for example a JSON object might include the **@** character, however, the Graphical Data Mapping editor allows only element names that are valid in XML or XPath.

3. Define transforms between the input elements and the output elements.
   - Use the **Assign** transform to set a fixed value without the use of input data.
   - Use the **Create** transform to create an empty JSON object, or a NULL JSON object, without the use of input data
   - Use the **Move** transform to copy an input element value to the output element.
   - Use the **If** transform, a custom transform, or the **XPath** transform to define the condition that determines the value of an output element.
   - Use the **Append** transform, the **Group** transform, the **For Each** transform, the **Join** transform, a custom transform, or the **XPath** transform to set the value of a JSON array.
   - Use the **Select** transform or the **Database Routine** transform to set the value with data available in a database system.
   - Use the **xs:type** transform to cast the value of a simple input element to a specific data type.
   - Use the **fn:type** transform to set the value by using XPATH 2.0 functions.
   - Use the **Custom Java** transform to enter your own Java code.
   - Use the **Custom ESQL** transform to call your own ESQL code.
   - Use a **Custom XPath** transform with the function `iib:nullValue()` to set a JSON object to NULL, or a JSON array to NULL.

   For more information, see “Specifying a transform (mapping operation)” on page 1430.

Example

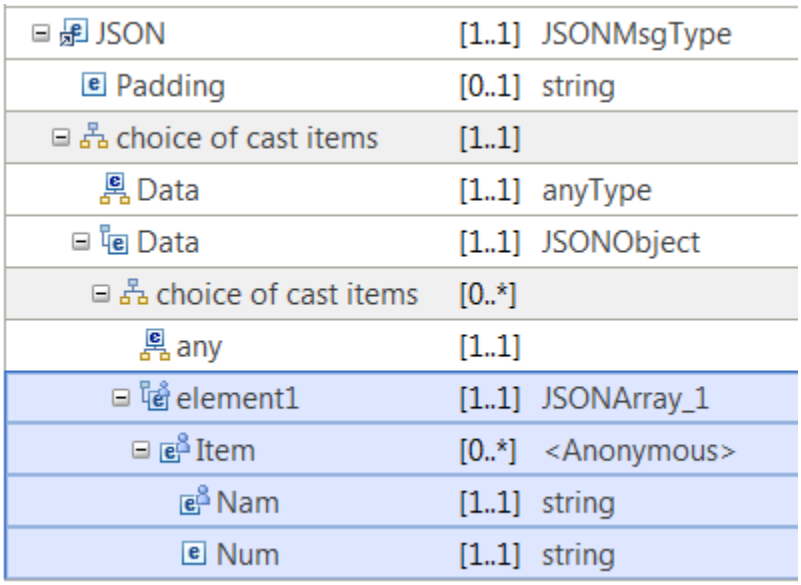
This example shows how to model in a message map the following JSON object message:

```
{"element1":[{"Nam":"val1","Num":1},{"Nam":"val2","Num":2}]}
```

To define this JSON message, you must

1. Add a user-defined element of type **JSONArray_Default**, and change its name to `Message`.
2. Set the type of **Item** to **Anonymous**.
3. Add two elements that represent the object-value elements in the array. The display name in the map is representative of the name of the object. At run time, the Graphical Data Mapping editor uses the name of the objects that are passed in the message.

The following figure shows the map after you complete the previous steps:

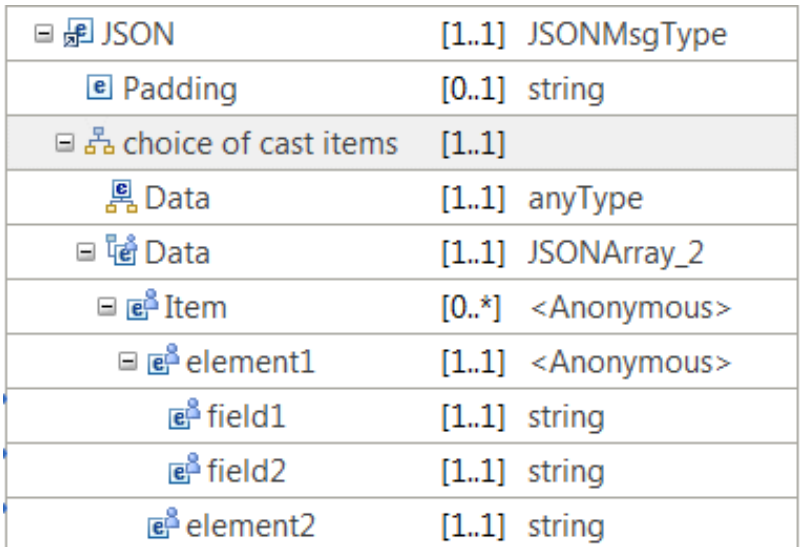


The message tree for the JSON message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x2f030fa0]
   (0x01000000:Object):Data = (
     (0x01001000:Array):element1 = (
       (0x01000000:Object):Item = (
         (0x03000000:NameValue):Nam = 'val1' (CHARACTER)
         (0x03000000:NameValue):Num = 1 (INTEGER)
       )
       (0x01000000:Object):Item = (
         (0x03000000:NameValue):Nam = 'val2' (CHARACTER)
         (0x03000000:NameValue):Num = 2 (INTEGER)
       )
     )
   )
 )
```

This other example shows how to model in a message map the following JSON array message:

```
[{"element1":{"field1":"Sweet","field2":"Flower"},"element2":"me@bbloggs.com"}]
```



The message tree for the JSON array message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x292051a0]
    (0x01001000:Array):Data = (
      (0x01000000:Object):Item = (
        (0x01000000:Object   ):element1 = (
          (0x03000000:NameValue):field1 = 'Sweet' (CHARACTER)
          (0x03000000:NameValue):field2 = 'Flower' (CHARACTER)
        )
        (0x03000000:NameValue):element2 = 'me@bbloggs.com' (CHARACTER)
      )
    )
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see “Troubleshooting a message map” on page 1587.

Graphically modeling a JSON object in a message map
In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to create a JSON object.

Before you begin

Create a message map with the output domain set to JSON. For more information, see "Graphically modeling a JSON message in a message map" on page 1554

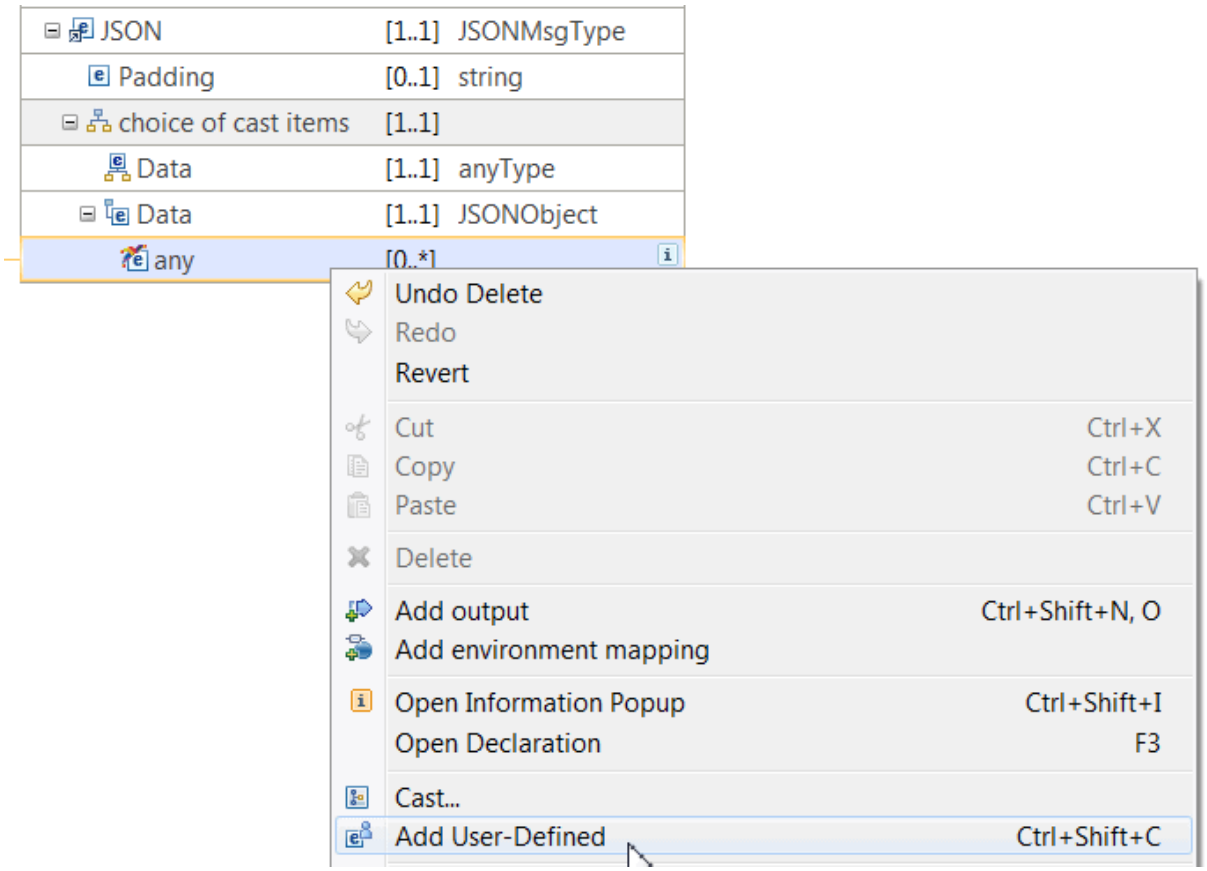
About this task

If you have a JSON schema for your message data, you can use that as described in "Creating or transforming a JSON message by using a JSON schema" on page 1547. Alternatively, you can model a JSON object in your message map by following these steps:

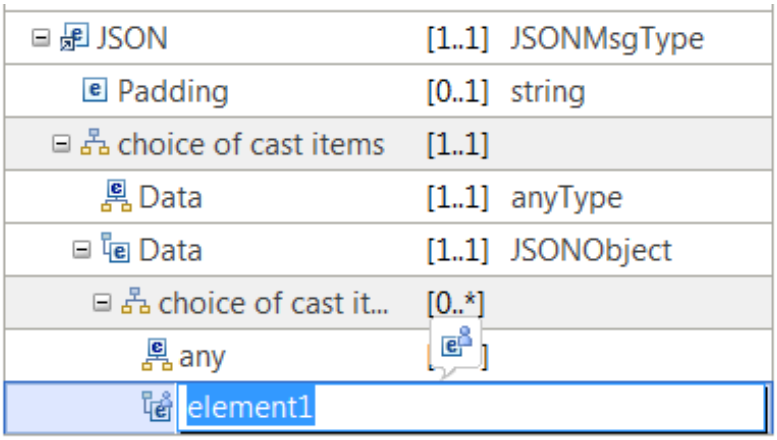
Procedure

To create a JSON object message with one name-value pair, complete the following steps:

1. Right-click **any**, and select **Add User-Defined**.



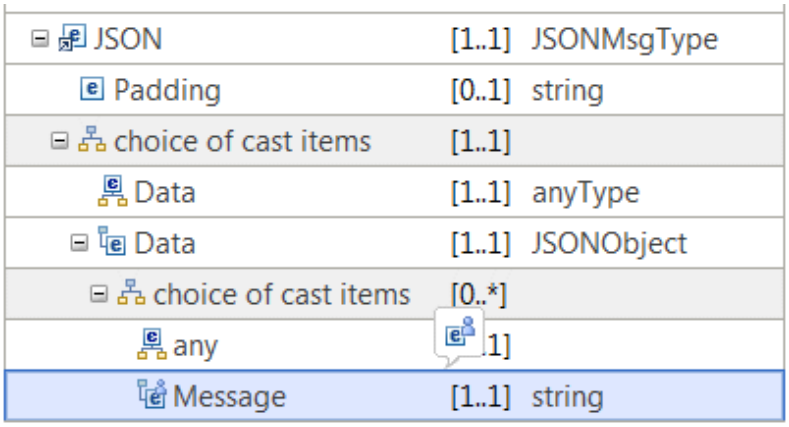
You can see **element1** created under the **any** element.



2. Click **element1** and enter the name of the JSON element.

   For example, enter **Message** for the following JSON message:

```
{"Message":"Hello World"}
```



3. Select the data type of the JSON element. You can select any of the following simple data types:

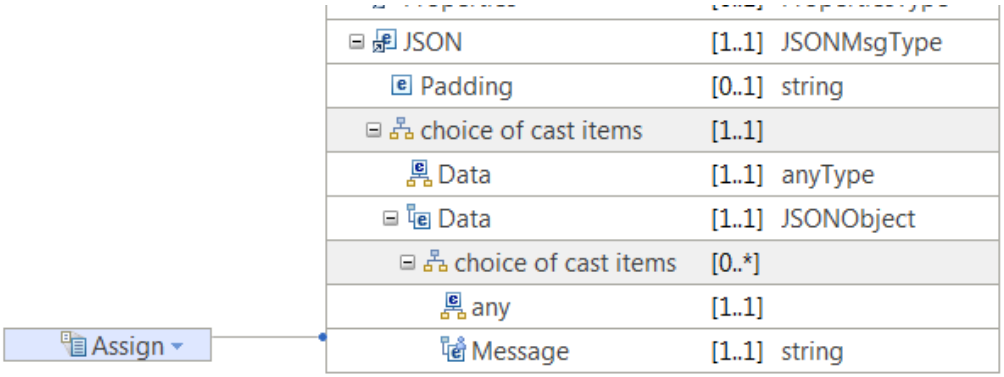
   - Anonymous: Use this type to define a JSON object.
   - Boolean
   - date
   - dateTime
   - decimal
   - double
   - duration
   - float
   - hexBinary
   - int
   - long
   - string
   - time
   - Global type, that is, a type based on an XML schema, a DFDL schema, or a message set. (You access your global types by selecting the option **Browse**.)

   To set the type, select the user-defined element. Then, complete one of the following steps:

   - In the map, click the type of the user-defined element to open the **Type Selection** window. Then, choose the type of the element.
   - In the **Properties** page, select the properties page drop-down to choose the type of the element.
   - In the **Properties** page, click **Browse** to open the **Type Selection** window. Then, choose the type of the element.

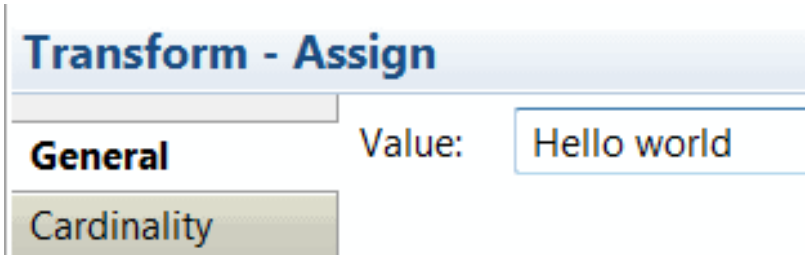
   When you change the type to **Anonymous**, the Graphical Data Mapping editor adds a child element.

4. Select a transform. For example, use an **Assign** transform to set the fixed value of a simple element.



5. Set the value of the element in the Properties view. In the **General** tab, enter a value in the **Value** property.

   For example, enter `Hello world` to set the value of the element Message.



Results

You create a simple JSON object message with one name-value pair.

The message tree for the following JSON message

```
{"Message":"Hello world"}
```

is described in the following trace:

```
Message: ( ['json' : 0xc552990]
  (0x01000000:Object   ):Data    = (
    (0x03000000:NameValue):Message = 'Hello World' (CHARACTER)
 )
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see “Troubleshooting a message map” on page 1587.

Graphically modeling a JSON array in a message map
In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to create a JSON array in a JSON message. You can create JSON arrays where all occurrences of the array are of the same type.

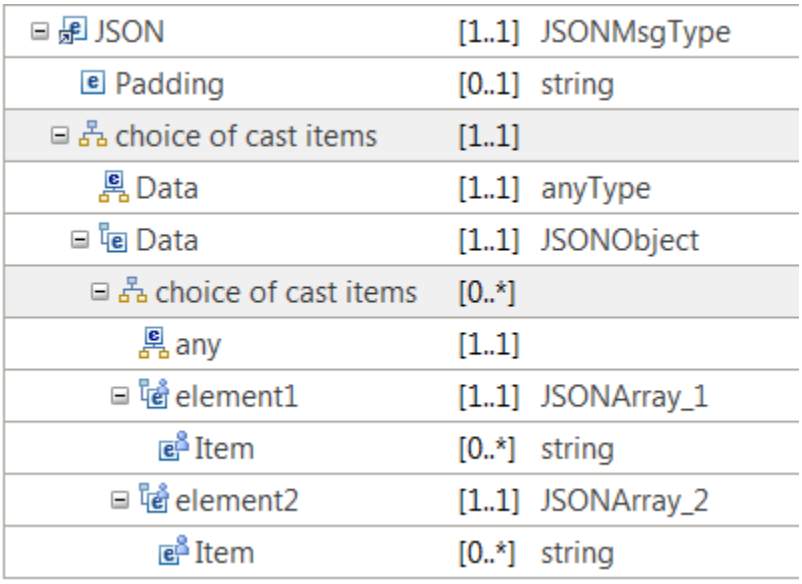
Before you begin

Create a message map with the output domain set to JSON. For more information, see “Graphically modeling a JSON message in a message map” on page 1554

About this task

To create a JSON array in a JSON message by using the **Add User-Defined** function, you must set the type of the user-defined element to **JSONArray_Default** and then set the type of the **Item** element to the required type.

When you define multiple arrays in your map, each JSON array has a type **JSONArray_N**, where **N** describes the unique id of the array in the map.

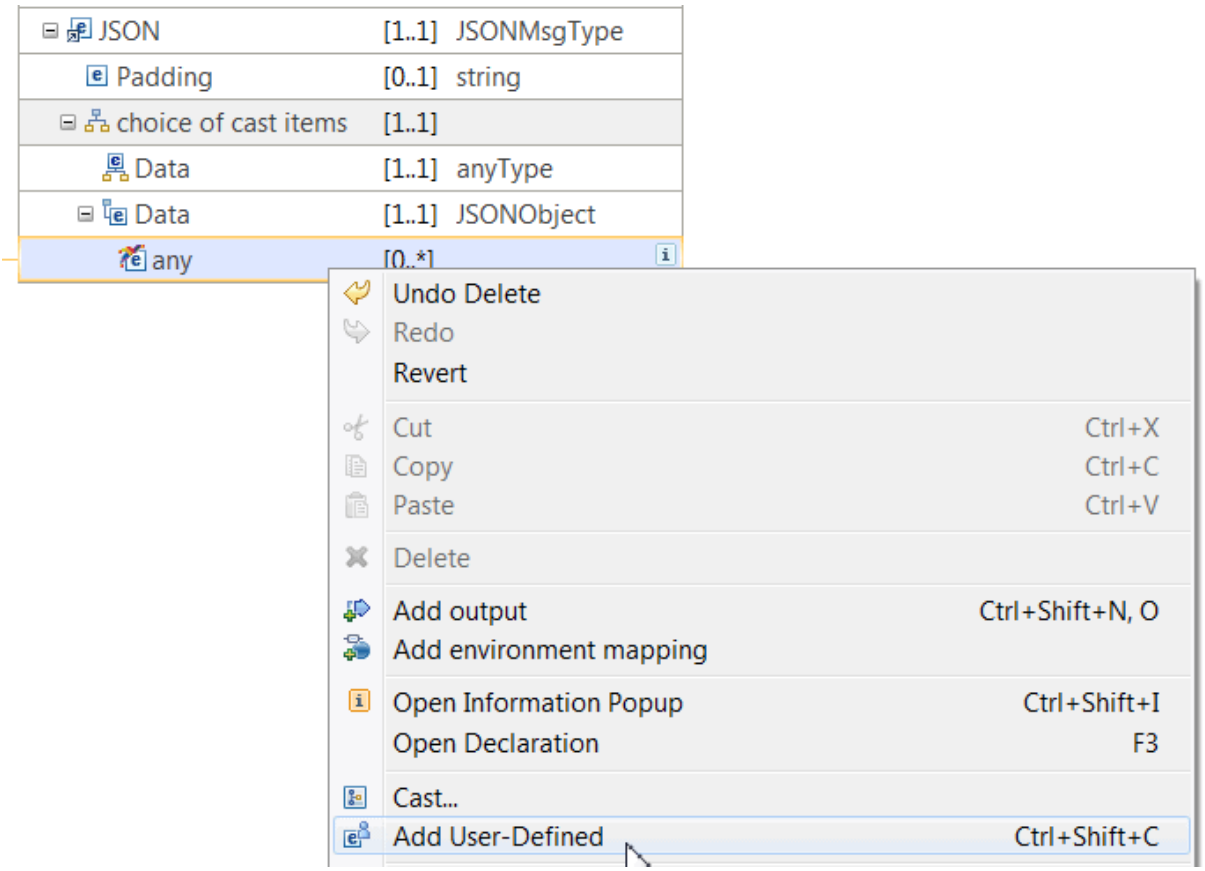


You can reuse the definition of a JSON array in your message map. For more information, see "Reusing a JSON message, a JSON object, or a JSON array in a message map" on page 1572.

Procedure

Complete the following steps to create a JSON array in a JSON message:

1. Right-click **any**, and select **Add User-Defined**.



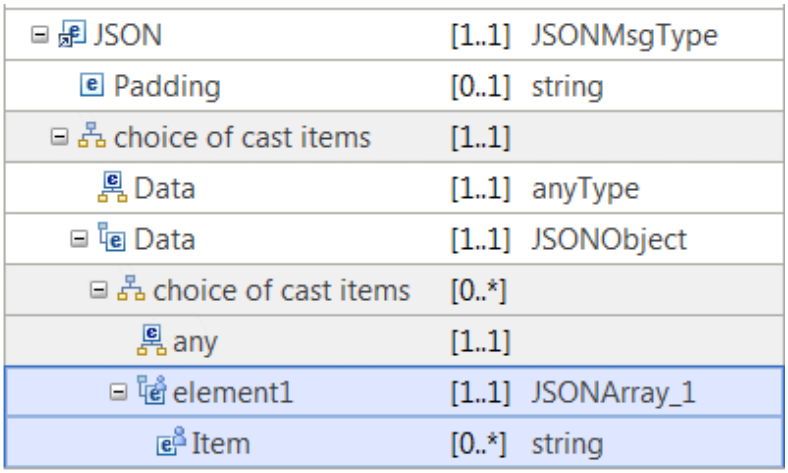
   The element **element1** is created under the **any** element.
2. Click **element1** and enter the name of the JSON array.

   By default, the user-defined element is created as a **string** element.
3. For **element1**, click **string**, and select **JSONArray_Default** as its type.

   The repeating element **Item** is created automatically. **Item** is used internally by IBM App Connect Enterprise. You cannot change this name. **Item** will not appear in your JSON data. Item is the logical

tree representation of each instance in the JSON array. For more information, see JSON parser and domain.

You cannot change the cardinality of **Item**. For more information, see "Configuring the cardinality of a user-defined element" on page 1412.



Note: **Item** is the name of the repeating element that describes the JSON array. All the values in the array have the same type. You set the type of **Item**. The first and the last element in the array is of the same type.

4. Use any of the following transforms to set the value of a JSON array:
   - Use the **For Each** transform to set the JSON array output value when you have a repeating input element. For more information, see "For Each" on page 1320.
   - Use the **Append** transform to add occurrences of the array in the order of the inputs to the transform, which can be singleton or repeating. For more information, see "Append" on page 1285.
   - Use the **Join** transform when you have multiple repeating input elements that you must combine into a single repeating output element. For more information, see "Join" on page 1332.

Example

This sample shows how to model in a message map the following JSON array. The JSON array is created as an element of a JSON object message:

```
{"Message":["valueA","valueB"]}
```

To define this JSON message, you must

1. Add a user-defined element of type **JSONArray_Default**, and change its name to `Message`.
2. Set the type of **Item** to string.

The following figure shows the map after you complete the previous steps:

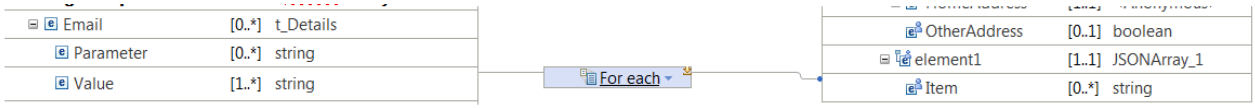
JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONObject
choice of cast items	[0..*]	
any	[1..1]	
Message	[1..1]	JSONArray_1
Item	[0..*]	string

The message tree for the JSON array in a JSON message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x2f0312d0]
    (0x01000000:Object):Data = (
      (0x01001000:Array):Message = (
        (0x03000000:NameValue):Item = 'valueA' (CHARACTER)
        (0x03000000:NameValue):Item = 'valueB' (CHARACTER)
       )
     )
)
```

When you use a **For Each** transform to set the value of the array elements, notice the difference of how the output message is created. The output differs based on how you connect the input and output elements. In this example, you can see how to transform an input message with two email addresses.

When you wire the transform to the element **Item** of type **string**, you get one array with multiple output elements:



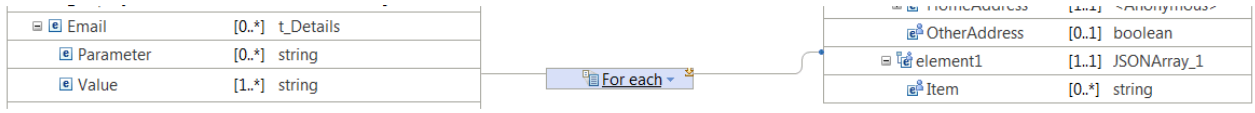
The following code is the message map representation of the JSON array as shown in the previous figure:

```
    (0x01001000:Array    ):element1      = (
       (0x03000000:NameValue):Item = 'myfirstemail@bbloggs.com' (CHARACTER)
       (0x03000000:NameValue):Item = ' mysecondemail@bbloggs.com' (CHARACTER)
     )
```

The JSON array that is created has two elements:

```
"element1":["myfirstemail@bbloggs.com"," mysecondemail@bbloggs.com"]
```

When you wire the transform to the element **element1** of type **JSONArray_1**, you get two arrays with the same name in the output:



The following code is the message map representation of the JSON array as shown in the previous figure:

```
 (0x01001000:Array    ):element1      = (
       (0x03000000:NameValue):Item = 'me@bbloggs.com' (CHARACTER)
     )
     (0x01001000:Array    ):element1      = (
```

```
        (0x03000000:NameValue):Item = ' mysecond@bbloggs.com' (CHARACTER)
      )
```

The JSON elements that are created are two arrays with the same name:

```
"element1":["myfirstemail@bbloggs.com"],"element1":[" mysecondemail@bbloggs.com"]
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see "Troubleshooting a message map" on page 1587.

Graphically modeling a multidimensional JSON array in a message map
In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to create a multidimensional JSON array, also known as a nested JSON array.

Before you begin

Create a message map with the output domain set to JSON. For more information, see "Graphically modeling a JSON message in a message map" on page 1554

About this task

If you have a JSON schema for your message data, you can use that as described in "Creating or transforming a JSON message by using a JSON schema" on page 1547. Alternatively, you can model a multidimensional JSON array in your message map by following these steps:

Procedure

To create a multidimensional JSON array in a JSON object message, complete the following steps:

1. Right-click **any**, and select **Add User-Defined**.

   The element **element1** is created under the **any** element.
2. Click **element1** and enter the name of the JSON array.

   By default, the user-defined element is created as a **string** element.
3. For **element1**, click **string**, and select **JSONArray_Default** as its type.

   The repeating element **Item** is created automatically. You cannot change this name. Item is the logical tree representation of each instance in the JSON array.
4. Set the type of **Item** to **JSONArray_Default**.

   A second **Item** element is created automatically.

JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONObject
choice of cast items	[0..*]	
any	[1..1]	
element1	[1..1]	JSONArray_1
Item	[0..*]	JSONArray_2
Item	[0..*]	string

5. Set the type of the nested **Item** element.

   You can use any of the following types:

   - Anonymous: Use this type to define a complex structure.
   - Boolean
   - date
   - dateTime
   - decimal
   - double
   - duration
   - float
   - hexBinary
   - int
   - long
   - string
   - time
   - JSONArray_Default: Use this type to define nested JSON arrays.
   - Global type, that is, a type based on an XML schema, a DFDL schema, or a message set. (You access your global types by selecting the option **Browse**.)
6. Use any of the following transforms to set the value of a JSON array:
   - Use the **For each** transform to set the JSON array output value when you have a repeating input element.
   - Use the **Append** transform to add occurrences of the array in the order of the inputs to the transform, which can be singleton or repeating.
   - Use the **Join** transform when you have multiple repeating input elements that you must combine into a single repeating output element.

Example

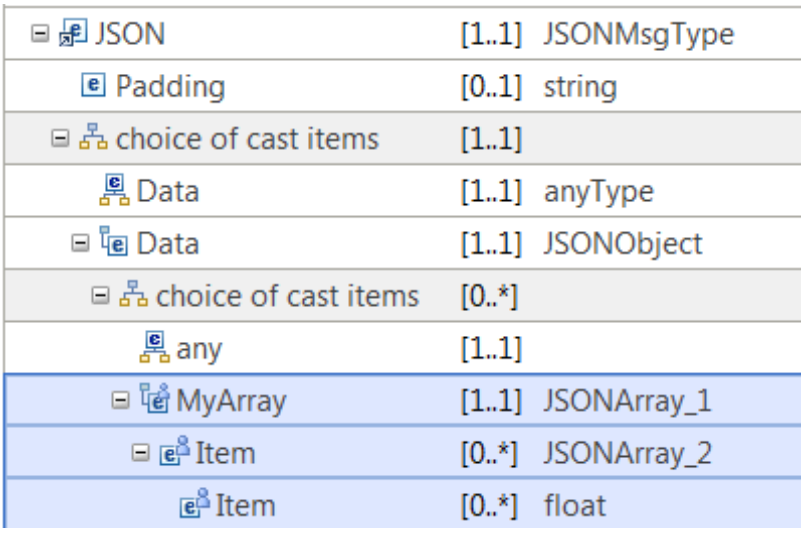
This sample shows how to model in a message map a two-dimensional JSON array in a JSON object message:

```
{"MyArray":[ [1.1], [2.1, 2.2] ]}
```

To define this JSON message, you must

1. Add a user-defined element of type **JSONArray_Default**, and change its name to `MyArray`.
2. Set the type of **Item** to **JSONArray_Default**.
3. Set the type of the nested **Item** element to `float`.

The following figure shows the map after you complete the previous steps:



The message tree for the JSON message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x2f033910]
   (0x01000000:Object):Data = (
     (0x01001000:Array):MyArray = (
       (0x01001000:Array):Item = (
         (0x03000000:NameValue):Item = 1.1E+0 (FLOAT)
       )
       (0x01001000:Array):Item = (
         (0x03000000:NameValue):Item = 2.1E+0 (FLOAT)
         (0x03000000:NameValue):Item = 2.2E+0 (FLOAT)
       )
     )
   )
 )
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see “Troubleshooting a message map” on page 1587.

Transforming heterogeneous JSON data in a message map
In the Graphical Data Mapping editor, you can use the **Add User-Defined** function to transform heterogeneous JSON data.

Before you begin

Create a message map with the output domain set to JSON. For more information, see “Graphically modeling a JSON message in a message map” on page 1554

About this task

The approach for modeling JSON heterogeneous data is to define each structure as a union of the different data that can be present in each possible form. Then, set the data, which is present in some instances of the data, as optional.

By default, when you model JSON data in the Graphical Data Mapping editor, you create objects and arrays where all instances are the same. All objects with the same name have the same set of data fields, and all entities in an array have the same set of data fields. This use case matches the most common usage of JSON data formats.

JSON allows entries in JSON arrays to have different structure and types. For JSON Objects, some JSON applications may also allow the same name for contained objects or values which may then have different structure and types. The JSON specification states: "The names within an object SHOULD be unique." For

more information, see The JavaScript Object Notation (JSON) Data Interchange Format). Only use this format if it is required by the external applications. This format of JSON data is termed heterogeneous JSON data.

If you have a JSON schema for your message data, you can use that as described in "Creating or transforming a JSON message by using a JSON schema" on page 1547. Alternatively, you can model heterogeneous JSON data by using the **Add User-Defined** function as described in this topic, or you can use an equivalent XML schema model as described in "Modeling a JSON message for use in a message map by using an equivalent XML schema model" on page 1572.

In the Graphical Data Mapping editor, the following rules apply when you model a heterogeneous JSON array:

- All entries in the array must contain either objects or values.
- You can model heterogeneous JSON data object arrays by using the **Add User-Defined** function.
- To model heterogeneous JSON value arrays, where the type of each instance can vary, you must use an external schema to provide the type of the **Item** element that represents each value entry in the array.

Procedure

Complete the following steps to model heterogeneous JSON data in the Graphical Data Mapping editor:

1. Build a union of the possible data fields, and set the elements that do not exist in every instance as optional.
   a) Use the **Add User-Defined** function to add user-defined elements to the map.
   b) In the **Properties** page of each element, configure the element's **General** tab to set the minimum occurrence to **0** so that the element is optional.

   Alternatively, you can use an external schema to provide the type of the **Item** element that represents each value entry in the array. The type can be defined as a schema union of simple types or an **xsd:anySimpleType**.
2. To model a JSON object with the same named but different typed contained objects, add the object once and then set the **maximum occurrence** of its cardinality to indicate the number of occurrences. The single object is a union of the possible content.
3. Use conditional mappings to read or construct the output data in each instance. For more information, see "Choosing a transform to define a conditional mapping" on page 1367.

   When you map to a heterogeneous JSON value modeled as a union of possible types, you must ensure the input data or value that is returned from the transform is of the type that is required for the instance being set. You can use the **xs:<type>** cast functions to set the data type that is required.

Example

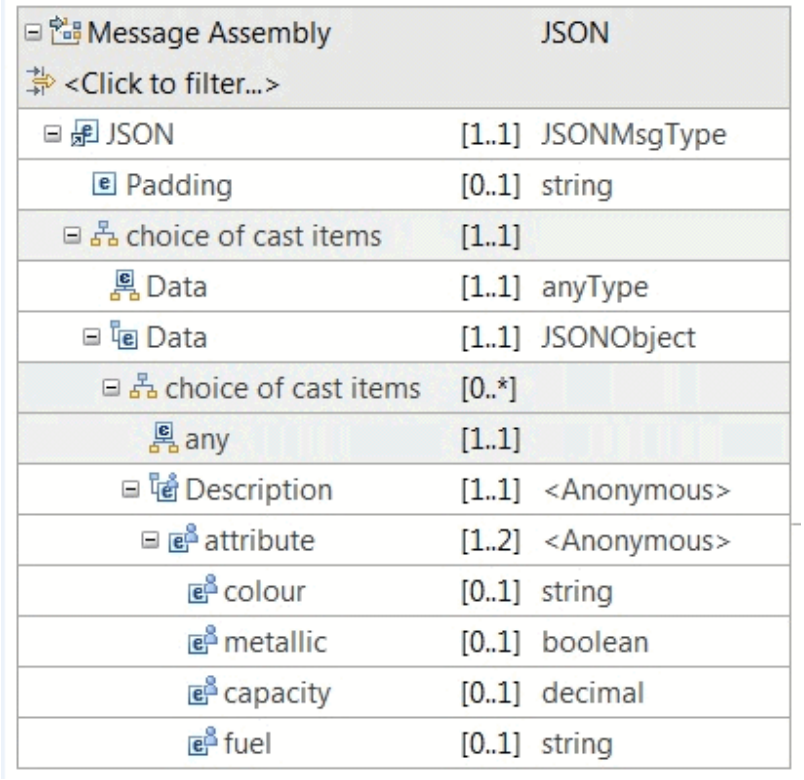
This example shows how to transform a JSON object message that describes a vehicle that uses two heterogeneous JSON data objects with different named values into a heterogeneous JSON array message where each entry is a data object with different named values.

The input application produces a heterogeneous JSON object message that describes a vehicle that uses two heterogeneous JSON data objects. Each JSON data object is called **attribute** and contains different name-value elements. The input JSON Data is as below:

```
{ "Description" : {
      "attribute"  : {
          "colour" : "blue",
          "metallic" : true
      },
      "attribute"  : {
          "capacity" : 1.4,
          "fuel" : "diesel"
      }
  }
}
```

In the map, the **Description** object is modeled with an attribute object that can occur twice (**maximum occurrence** set to be **2**).

- The heterogeneous attribute object contains the name - values **colour** and **metallic** or **capacity** and **fuel**.
- Each of the name - values is modeled as optional (**minimum occurrence** set to be **1**).



The target application requires this information as a heterogeneous JSON array:

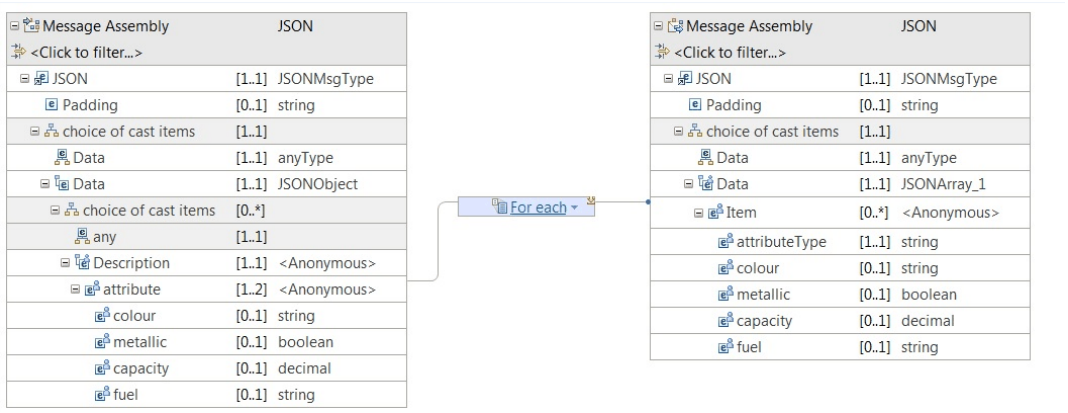
```
[{"attributeType":"Paint","colour":"blue","metallic":true},
{"attributeType":"Engine","capacity":1.4E+0,"fuel":"diesel"}]
```

In the map, the output JSON array is defined as the union of the possible data fields:

- The array is represented by a JSON object of type **Anonymous**.
- The first element is a single required element **attributeType**. All entries in the array contain this name-value.
- The other name-value elements are **colour**, **metallic**, **capacity**, and **fuel**. These elements do not occur in all entities in the array. All these elements are defined as optional. Each element has its property **minimum occurrence** set to be **0**.

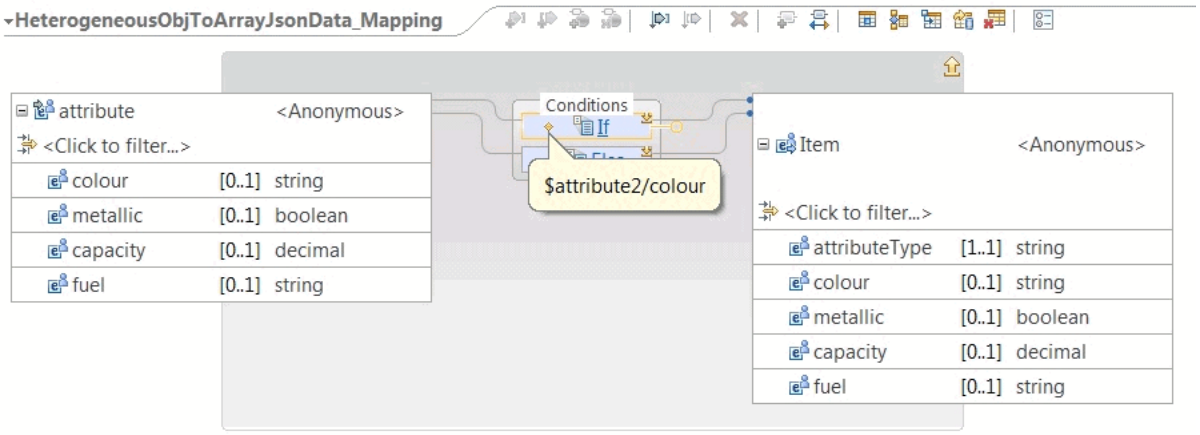
JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONArray_1
Item	[0..*]	<Anonymous>
attributeType	[1..1]	string
colour	[0..1]	string
metallic	[0..1]	boolean
capacity	[0..1]	decimal
fuel	[0..1]	string

To transform from the input to the required output, you use the **For Each** transform to create each of the entries in the array from the same named (repeating) attribute input objects.

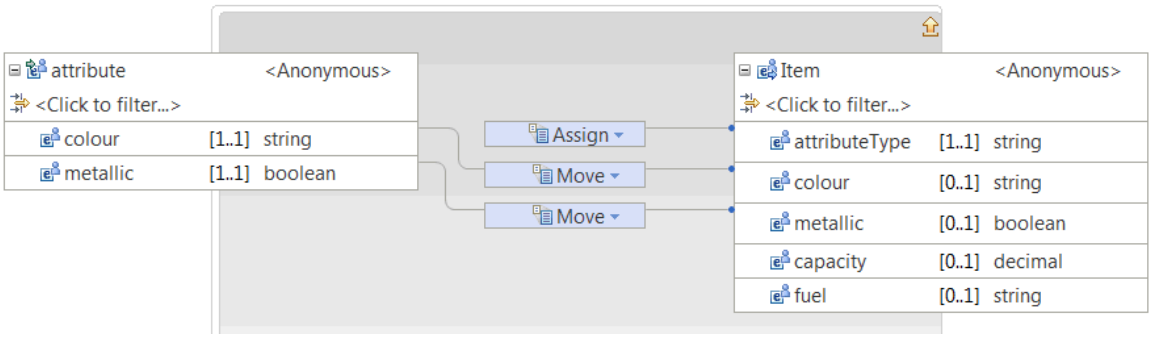


Inside the nested map associated with the **For Each** transform, you define an **If** transform that checks whether the **attribute** element contains **colour** or **capacity**.

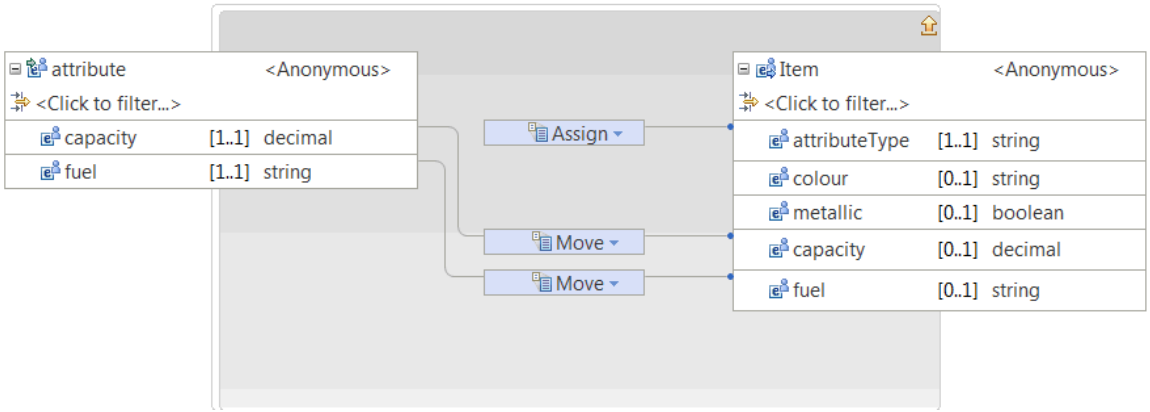
1. You define the **If** condition to check for **colour**: `$attribute2/colour` When this expression evaluates to true, the transformation in the nested map associated with the **If** is applied.
2. The **Else** path does not require a condition. When the **If** expression evaluates to false, the transformation in the nested map associated with the **Else** is applied.



The nested map that is associated to the **If** transform sets the **attributeType** element to **Paint** and copies the value of the other elements unchanged.



The other nested map that is associated to the **Else** sets the **attributeType** element to **Engine** and copies the value of the other elements unchanged.



When you run this map, you get the following JSON message:

```
[{"attributeType":"Paint","colour":"blue","metallic":true},
{"attributeType":"Engine","capacity":1.4E+0,"fuel":"diesel"}]
```

You obtain the following output tree structure:

```
(0x01000000:Object):JSON       = ( ['json' : 0x298f1e00]
    (0x01001000:Array):Data = (
      (0x01000000:Object):Item = (
        (0x03000000:NameValue):attributeType = 'Paint' (CHARACTER)
        (0x03000000:NameValue):colour        = 'blue' (CHARACTER)
        (0x03000000:NameValue):metallic      = TRUE (BOOLEAN)
      )
```

```
      (0x01000000:Object):Item = (
        (0x03000000:NameValue):attributeType = 'Engine' (CHARACTER)
        (0x03000000:NameValue):capacity      = 1.4E+0 (FLOAT)
        (0x03000000:NameValue):fuel          = 'diesel' (CHARACTER)
      )
    )
  )
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see "Troubleshooting a message map" on page 1587.

Reusing a JSON message, a JSON object, or a JSON array in a message map
In the Graphical Data Mapping editor, you can reuse JSON objects and JSON arrays that you define by using the **Add User-Defined** function. You can also define reusable JSON structures with an XML schema.

About this task

You can reuse JSON objects and JSON arrays between message maps or between message assemblies.

Procedure

You can use any of the following methods to define a JSON message, a JSON object, or a JSON array that you plan to reuse in a map:

- Define a JSON message, a JSON object, or a JSON array by using the **Add User-Defined** function. Then, choose any of the following methods to reuse an existing JSON definition:
  - Use copy and paste functions available in the map through the menu.
  - Use the **Browse** functionality that is available when you change the type of an element in your map.

  For more information, see "Graphically modeling a JSON message in a message map" on page 1554, "Graphically modeling a JSON object in a message map" on page 1557 and "Graphically modeling a JSON array in a message map" on page 1560.
- Define a JSON message, a JSON object or a JSON array with an XML schema. Then, use the **Cast** function to reuse the JSON object or JSON array definition.

  For more information, see "Modeling a JSON message for use in a message map by using an equivalent XML schema model" on page 1572.

What to do next

Deploy the message map and verify that the output message is valid. For more information, see "Troubleshooting a message map" on page 1587.

Modeling a JSON message for use in a message map by using an equivalent XML schema model
If you do not have a JSON schema for your JSON message data, you can use the following procedure to create an equivalent XML schema model that can be used in one or more message maps.

About this task

Graphical data mapping requires a supported message model of your message data.

If you have a JSON schema for your message data, you can use it as described in "Creating or transforming a JSON message by using a JSON schema" on page 1547. If you do not have a JSON schema, you can model a JSON message that contains JSON objects, JSON arrays, or both, by following the steps in this topic to create an equivalent XML schema model, which you can then use in one or more message maps with the `Cast` function in the Graphical Data Mapping editor. For more information, see "Graphically modeling a JSON message in a message map" on page 1554.

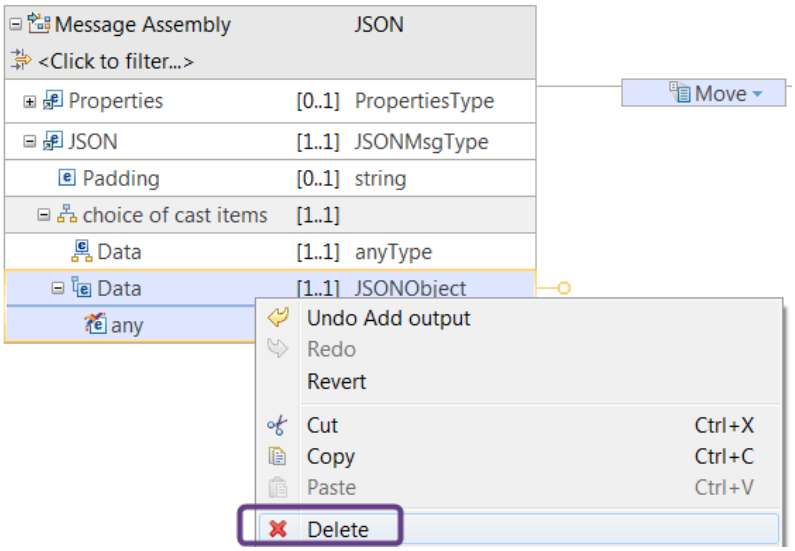
The following steps detail how to use an XML schema model in one or more message maps to define the JSON data from a corresponding XML schema type. Alternatively, you can define the JSON data in-line in the map by using the **Add User-Defined** function.

Procedure

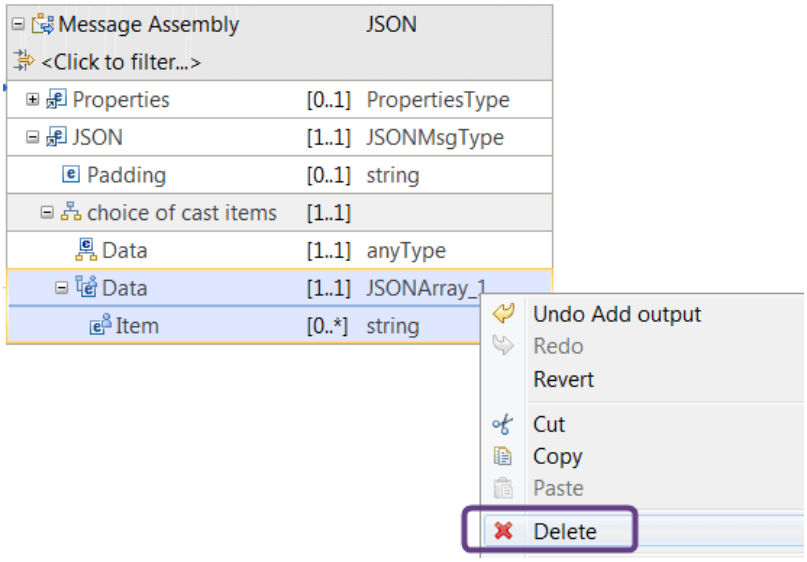
Complete the following steps to create a JSON message based on a schema model:

1. Create a message map. Select one of the following JSON supplied models as the output of the map:
   - **JSON (JSON object message model supplied by IBM)** to create a JSON object message
   - **JSON (JSON array message model supplied by IBM)** to create a JSON array message

   For more information, see "Creating a message map" on page 1382.
2. Delete the predefined **Data** object. Choose one of the following options:
   - If you select **JSON (JSON object message model supplied by IBM)** in the previous step, delete the **Data** element of type **JSONObject**.



   - If you select **JSON (JSON array message model supplied by IBM)** in the previous step, delete the **Data** element of type **JSONArray_1**.



Note: In the Graphical Data Mapping editor, you can only have one **Data** element in a JSON message if you use a schema model to define the JSON message.

3. Provide a definition of the required JSON message by using the **Cast** function to redefine the **Data** element from anyType to your schema type.

   For more information, see "Casting elements in a message map" on page 1398.

   When you cast the *JSON.Data* element to define the JSON message structure, you can only use global schema types from your schema models. For the input side of the map, the global schema type must be defined in the default namespace.

   For example, if you build a JSON input with a cast to an element in a none empty namespace, the Mapping node runs without throwing an error. However, you might not get the expected output. You can use a user trace to see that the presence of the namespace can be deduced as the cause of the map failure:

   a. First, you see that the map successfully locates JSON.Data.

   b. Next, you see BIP message `BIP3959I: The Mapping node is traversing the input tree by using the nodetest ''element(Data)'' and the relationship ''child''. The number of matching elements is ''1''.`

   c. Finally, you see that the map fails to locate anything for the subsequent JSON objects. You see BIP message `BIP3959I: The Mapping node is traversing the input tree by using the nodetest ''element({http://schema014_T}:testMsgNsImpIn)'' and the relationship ''child''. The number of matching elements is ''0''.`

   When you cast the *JSON.Data.any* on the output side of the map to define the JSON message, you can use a global schema type from a non-default namespace. It is recommended that you use schema types in the default namespace because the Mapping node creates the elements with namespaces, but the JSON serializer then removes them.

   The Mapping node creates the elements with namespaces and the JSON serializer ignores them.

   Note: When you define a JSON message by using the **Cast** function, you must comply with the following rules:

   - You must define the schema definition for the type of a JSON object in the default namespace of the project where the map is defined.
   - You must define the schema definition for a JSON array in the target namespace: `http://www.ibm.com/iib/msl/json`.
   - You must use the following naming convention for the name of the type of a JSON array: `JSONArray_`*`unique_name`*, where *unique_name* is any combination of valid XML characters. For example, you can name a type as `JSONArray_ofBooleans` if the type is an array of Boolean elements. You can name a type `JSONArray_ofMyAddrObjects` if the type is a complex type.
   - The schema of a JSON array must contain a sequence with a single repeating element named `Item`.
   - You can only use global, not local, schema types from your schema models.
   - Optionally, you can define the JSON object or JSON array as nillable in the schema model to support the JSON null value.

4. Define transforms between the input elements and the output elements below `JSON.Data`. You must not wire any transform directly to the output `JSON.Data` element. You can map to its child elements only.
   - Use the **Assign** transform to set a fixed value without the use of input data.
   - Use the **Create** transform to create an empty element, or a nil element without the use of input data.
   - Use the **Move** transform to copy an input element value to the output element.
   - Use the **If** transform, a custom transform, or the **XPath** transform to define the condition that determines the value of an output element.
   - Use the **Append** transform, the **Group** transform, the **For Each** transform, the **Join** transform, a custom transform, or the **XPath** transform to set the value of a JSON array.
   - Use the **Select** transform or the **Database Routine** transform to set the value with data available in a database system.
   - Use the **xs:type** transform to cast the value of a simple input element to a specific data type.
   - Use the **fn:type** transform to set the value by using XPATH 2.0 functions.
   - Use the **Custom Java** transform to enter your own Java code.
   - Use the **Custom ESQL** transform to call your own ESQL code.

   For more information, see “Specifying a transform (mapping operation)” on page 1430.

Example

The following examples present XML schema types that model some example JSON data objects and arrays.

Example for a JSON object message

This example shows how to define a schema model for a JSON object message in the default namespace:

```
{ "Nam" : "A Name", "Num" : 1 }
```

The message tree for the JSON message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x2f030fa0]
    (0x01000000:Object):Data = (
          (0x03000000:NameValue):Nam = 'A Name' (CHARACTER)
          (0x03000000:NameValue):Num = 1 (INTEGER)
    )
  )
```

The following code shows the schema file that defines the JSON object message:

```
<?xml version="1.0" encoding="UTF-8"?>
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="MyJsonObjType1">
     <xsd:sequence>
      <xsd:element name="Nam" type="xsd:string" nillable="true"></xsd:element>
      <xsd:element name="Num" type="xsd:int" nillable="true"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

The following figure shows the map after you cast the **Data** element of the JSON object message to the schema model **MyJsonObjType1**:

JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	MyJsonObjType1
Nam	[1..1]	string
Num	[1..1]	int

Example for a JSON array message

This example shows how to model a JSON message that is an array of Boolean elements. You define a schema model for the following JSON array message in the default namespace:

```
[ true,false ]
```

The message tree for the JSON array message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x292051a0]
    (0x01001000:Array):Data = (
      (0x01000000:NameValue):Item  = TRUE (BOOLEAN)
      (0x01000000:NameValue):Item  = FALSE (BOOLEAN)
     )
)
```

The following code shows the schema file that defines the JSON array message in the target namespace **http://www.ibm.com/iib/msl/json**:

```
<?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  targetNamespace="http://www.ibm.com/
iib/msl/json">
     <xsd:complexType block="#all" name="JSONArray_ofBoolean">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="Item" nillable="true"
 type="xsd:boolean"/>
        </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

The following figure shows the map after you cast the **Data** element of the JSON object message to the schema model **JSONArray_ofBoolean**:

JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	JSONArray_ofBoolean
Item	[0..*]	boolean

Example for JSON object message with an array of objects

This example shows how to model a JSON message that is an array of objects.

```
[ { "nam":"one","val":1 },{ "nam":"two","val":2 }]
```

The message tree for the JSON object message is described in the following trace:

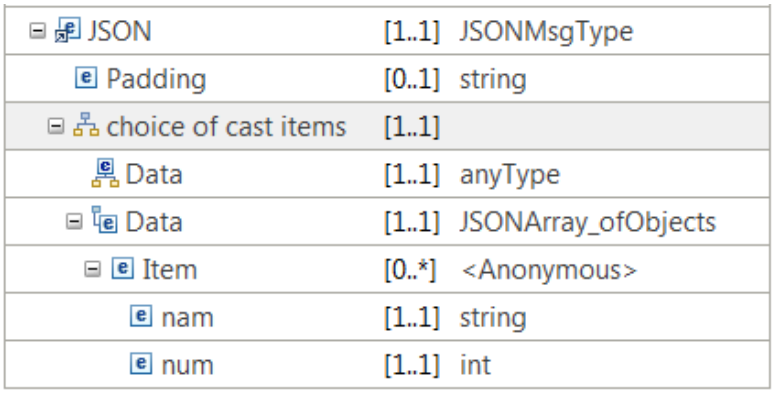
```
(0x01000000:Object):JSON       = ( ['json' : 0x292051a0]
    (0x01001000:Array):Data = (
                        (0x01000000:Object):Item = (
                                                   (0x03000000:NameValue):nam = 'one' (CHARACTER)

                                                   (0x03000000:NameValue):val = 1  (INTEGER)

                                                    )
                        (0x01000000:Object):Item = (
                                                   (0x03000000:NameValue):nam = 'two' (CHARACTER)

                                                   (0x03000000:NameValue):val = 2 (INTEGER)
                                                   )
                             )
)
```

You define a schema model for the JSON object message in the default namespace.

The following code shows the schema file that defines the JSON array. You define the **JSONArray_ofObjects** type in the target namespace **http://www.ibm.com/iib/msl/json**:

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
www.ibm.com/iib/msl/json">
         <xsd:complexType block="#all" name="JSONArray_ofObjects">
             <xsd:sequence>
                 <xsd:element maxOccurs="unbounded" minOccurs="0" name="Item" nillable="true">
                     <xsd:complexType>
                         <xsd:sequence>
                            <xsd:element name="nam" type="xsd:string" nillable="true"/>
                            <xsd:element name="num" type="xsd:int" nillable="true"/>
                         </xsd:sequence>
                    </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
 </xsd:schema>
```

The following figure shows the map after you cast the **Data** element of the JSON object message to the schema model **JSONArray_ofObjects**:



Example for JSON Object message containing a JSON array

This example shows how to model a JSON object message that contains a JSON array.

```
{ "state":"ok", "array":[ { "nam":"one","val":1 },{"nam":"two","val":2 } ] }
```

The message tree for the JSON object message is described in the following trace:

```
(0x01000000:Object):JSON       = ( ['json' : 0x292051a0]
    (0x01000000:Object):Data = (
```

```
             (0x03000000:NameValue):state= 'ok' (CHARACTER)
             (0x01001000:Array):array= (
                              (0x01000000:Object):Item = (
                                                  (0x03000000:NameValue):nam = 'one' (CHARACTER)

                                                  (0x03000000:NameValue):val = 1  (INTEGER)

                                                           )
                              (0x01000000:Object):Item = (
                                                  (0x03000000:NameValue):nam = 'two' (CHARACTER)

                                                  (0x03000000:NameValue):val = 2 (INTEGER)
                                                           )
                                             )
                               )
)
```

The object message is defined in the default namespace and imports the type of the JSON array element from the second schema. The following code shows the schema file that defines the JSON object message:

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:mjatns="http://www.ibm.com/
iib/msl/json">
      <xsd:import namespace="http://www.ibm.com/iib/msl/json"
 schemaLocation="MyJsonArrayTypes.xsd"/>
         <xsd:complexType name="MyJsonObjWithArrayType1">
             <xsd:sequence>
                 <xsd:element name="state" type="xsd:string" nillable="true"></xsd:element>
                 <xsd:element name="array" type="mjatns:JSONArray_ofObjects" nillable="true"></
xsd:element>
              </xsd:sequence>
         </xsd:complexType>
    </xsd:schema>
```

The following code shows the schema file that defines the JSON array. You define the **JSONArray_ofObjects** type in the target namespace **http://www.ibm.com/iib/msl/json**:

```
<?xml version="1.0" encoding="UTF-8"?>
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.ibm.com/
iib/msl/json">
       <xsd:complexType block="#all" name="JSONArray_ofObjects">
           <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="0" name="Item" nillable="true">
                                 <xsd:complexType>
                                           <xsd:sequence>
                                                 <xsd:element name="nam" type="xsd:string"
 nillable="true"/>
                                                 <xsd:element name="num" type="xsd:int"
 nillable="true"/>
                                           </xsd:sequence>
                                 </xsd:complexType>
                </xsd:element>
         </xsd:sequence>
     </xsd:complexType>
    </xsd:schema>
```

The following figure shows the map after you cast the **Data** element of the JSON object message to the schema model **MyJsonObjWithArrayType1**:

JSON	[1..1]	JSONMsgType
Padding	[0..1]	string
choice of cast items	[1..1]	
Data	[1..1]	anyType
Data	[1..1]	MyJsonObjWithArrayType1
state	[1..1]	string
array	[1..1]	JSONArray_ofObjects
Item	[0..*]	<Anonymous>
nam	[1..1]	string
num	[1..1]	int

Note: The XML schema can be extended if you need to use the message model in another IBM App Connect Enterprise component, for example a Route node or a monitoring event from a message flow node. For the map cast, you can define an XML schema type, for example `MyJsonObjType1`. If you want to use the schema type in the XPath expression builder, for setting a match expression for the Route node or the monitoring event data path, you add a schema element definition for the "Data" element of the JSON domain as follows:

```
<xsd:element name="Data" type="MyJsonObjType1"></xsd:element>
```

What to do next

Deploy the message map and verify that the output message is valid. For more information, see “Troubleshooting a message map” on page 1587.

Graphically modeling and transforming a JSONP message in a message map

You can use the Graphical Data Mapping editor to create or transform a JSONP object message or a JSONP array message. You can provide the model of your JSONP message in a JSON schema file in a shared library project. Alternatively, you can define your JSON data by using the **Add User Defined** function in the map.

About this task

A JSONP service, or Remote JSON service, is a web service that returns JSON data padded with a user-defined JavaScript function call.

When you configure a message map to transform a JSON message, the Graphical Data Mapping editor puts the JSON padding in the top-level **Padding** element, and the JSON data under the **Data** element. If JSON padding is detected by an input node, the name of the client-side script is placed in the top-level **Padding** element.

You can also use the Graphical Data Mapping editor to configure a message flow to provide a JSONP response. When a JSON message is serialized and the top-level **Padding** element has a value, the JSON message is wrapped in the provided client-side script name.

Procedure

Create or transform your JSONP data by completing one of the following tasks:

- Map your data by using a JSON schema. This option requires the JSON schema and map to be in the same shared library. For instructions on how to complete this task, see “Creating or transforming a JSON message by using a JSON schema” on page 1547.

- Map a user defined JSON object or a user defined JSON array. For instructions on how to complete this task, see "Creating or transforming a JSON output message by using the User Defined function to model the JSON data" on page 1552.

What to do next

Deploy and test the message map. For more information, see "Troubleshooting a message map" on page 1587.

Implementing a REST API operation by using a message map

You can use the Graphical Data Mapping editor to implement a REST API operation. When you create a map for a REST API operation, you can choose to have the map input and output defined automatically from the definitions in the REST API Swagger document or OpenAPI 3.0 document.

About this task

When you create the map for a Mapping node in a subflow that implements a REST API operation, you can select the option **Message map with input and output for REST API operation *operation_name***. When you select this option, the map is created with the input and output data fully defined by the JSON schema data definitions in the REST API Swagger document or OpenAPI 3.0 document.

The local environment is automatically added to the map input and any path or query parameters are added under **Local Environment** > **REST** > **Input** > **Parameters**, ready for you to connect transforms. A Task transform is pre-wired to **Local Environment** > **REST** > **Input** to help you locate this information.

The map input message body is populated according to the operation request body, and the output message body is populated according to the response body. When the body is defined as a model or JSON array, the map is created with a JSON domain message. When there is no body, or a simple type, the map is created with a BLOB domain message.

Note: If you update the REST API and change either the path or query parameters, or the request or response body, the message map is updated automatically to reflect the new definitions when it is reopened. If existing transforms were wired, they might become invalid or disconnected and the map requires modification to match the new data model. If the map was created when there was no request or response body defined for the operation, you must re-create the map to add one later.

Ensure that you are aware of the requirements for JSON schema with message maps. For more information, see "JSON schema requirements for message maps" on page 1549.

If you are implementing a REST API operation in which the processing can be achieved by using the "Transform types in the Graphical Data Mapping editor" on page 1282, you can implement the operation by using a single message map, as described in the following procedure. However, if the processing for your REST API operation requires that you transform to an intermediary data format and use additional nodes as part of your implementation, follow the procedure that is described in "Implementing a REST API operation with intermediary processing by using message maps" on page 1582.

Procedure

To implement an operation in a REST API by using a single message map, complete the following steps:

1. Create or open the subflow for the REST API operation.

   The REST API must be saved before a new message map for an operation can be created; this ensures that the required information in the Swagger document or the OpenAPI 3.0 document is current and consistent.
2. Add a Mapping node to the subflow.
3. Create the message map by double-clicking the Mapping node to open the **New Message Map** wizard. The wizard detects that the map is being created for the current REST API operation, and preselects the option **Message map with input and output for REST API operation *operation_name***. Click **Finish**.

4. The REST API operation message map is created in the REST API project and opened in the Graphical Data Mapping editor.
5. The input data from the REST API is provided in the map. This is located in the Local Environment, which is automatically added to the map input. Any REST API path or query parameters defined for the operation are automatically defined under **Local Environment** > **REST** > **Input** > **Parameters**.

   A **Task** transform is added to the map with its input connected to **Local Environment** > **REST** > **Input**, and enables you to locate the REST API input data that might be required in the mappings. You can remove the Task when you have completed your mappings. The map can be deployed and run with a **Task** transform, with no effect.

   Note: Input form and array type parameters are not supported.
6. The message body of the map input and output is also automatically defined by the REST API Swagger document or OpenAPI 3.0 documents.

   The input message body is defined as one of the following types:

   - JSON, with the data type of any request body defined for the operation in the Swagger document or OpenAPI 3.0 document.
   - BLOB, when the operation has no request body data

   The ouput message body is defined as one of the following types:

   - JSON, with the data type defined for the HTTP status 200 response. If there is no status 200 response, either the default response or the first response defined in the Swagger document or OpenAPI 3.0 document is used.
   - BLOB, when the operation has no response body data.

   The Data type of the JSON input or output is defined in the following way:

   - If the REST API operation request or response is defined as a model type (JSON reference to a type in the Swagger document or OpenAPI 3.0 document definitions) it has a type name as the model name; for example, `Pet` or `Department`. If it is an array of a type that is defined in the model, the type name is prefixed with `JSONArray_`; for example, `JSONArray_Pet`.
   - If the API operation request body is defined inline by a Swagger schema statement or an OpenAPI 3.0 schema statement, its type name is formed as:

     ```
     <Json type>_<operation name>_body
     ```
   - If the API operation response body is defined inline by a Swagger schema statement or an OpenAPI 3.0 schema statement, its type name is formed as:

     ```
     <Json type>_<operation name>_<http status code>
     ```

     For example:

     ```
     string_getSurname_200
     ```

     or

     ```
     object_getItem_200
     ```
   - If the request or response body is defined as an inline array, the type name is prefixed with `JSONArray_`; for example, `JSONArray_string_getAllSurnames_200`

7. Complete the required transformation logic in the map to implement the REST API operation, using any of the "Transform types in the Graphical Data Mapping editor" on page 1282. For more information, see "Editing message maps" on page 1394.

   In addition to transforming the message data, you can interact with resources as described in the following topics:

   - "Mapping database content" on page 1523
   - Using Java in a message map
   - "Accessing user-defined properties from a Mapping node" on page 1479

What to do next

Deploy the REST API with the message map and exercise the operation to verify that the output message is as you expected. For more information, see "Troubleshooting a message map" on page 1587.

Implementing a REST API operation with intermediary processing by using message maps

You can use the Graphical Data Mapping editor to implement a REST API operation. When you create a map for a REST API operation, you can choose to have the map input and output defined automatically from the definitions in the REST API Swagger document or OpenAPI 3.0 document.

About this task

When you create the map for a Mapping node in a subflow that implements a REST API operation, you can select the option **Message map with input and output for REST API operation *operation_name***. When you select this option, the map is created with the input and output data fully defined by the JSON schema data definitions in the REST API Swagger document or OpenAPI 3.0 document.

The local environment is automatically added to the map input and any path or query parameters are added under **Local Environment** > **REST** > **Input** > **Parameters**, ready for you to connect transforms. A Task transform is pre-wired to **Local Environment** > **REST** > **Input** to help you locate this information.

The map input message body is populated according to the operation request body, and the output message body is populated according to the response body. When the body is defined as a model or JSON array, the map is created with a JSON domain message. When there is no body, or a simple type the map is created with a BLOB domain message.

Note: If you update the REST API and change either the path or query parameters, or the request or response body, the message map is updated automatically to reflect the new definitions when it is reopened. If existing transforms were wired, they might become invalid or disconnected and the map will require modification to match the new data model. If the map was created when there was no request or response body defined for the operation, you must re-create the map to add one later.

Ensure that you are aware of the requirements for JSON schema with message maps. For more information, see "JSON schema requirements for message maps" on page 1549.

The following procedure describes the steps that you need to follow if you are implementing a REST operation that involves transforming to an intermediary data format as part of its processing. This is the case if your flow contains nodes that use a different message model from the one that is defined in the REST API. In this situation, you need to use a Mapping node at the beginning of the REST API operation subflow and another Mapping node at the end of the subflow. The first Mapping node takes the REST input from the Swagger document or OpenAPI 3.0 document and transforms it to an intermediary format to be used by your processing nodes. At the end of the subflow, the second Mapping node is required to transform from the intermediary format to the format required by the REST API operation response.

To summarize the sequence of events:

1. Transform from the message format of the REST API operation request to the format of the intermediary processing input (format *X*)
2. Process data using message format *X* and, when it is completed, return the format of the intermediary processing result (format *Y*)

3. Transform from message format *Y* to the format required by the REST API operation response.

Note: If you are implementing a REST API operation in which all the processing can be achieved by using the "Transform types in the Graphical Data Mapping editor" on page 1282, you can use a single message map, by following the procedure described in "Implementing a REST API operation by using a message map" on page 1580.

Procedure

To implement a REST API operation that involves intermediary processing to transform between data formats, complete the following steps:

1. Create or open the subflow for the REST API operation.

   The REST API must be saved before a new message map for an operation can be created; this ensures that the required information in the Swagger document or OpenAPI 3.0 document is current and consistent.
2. Add a Mapping node to the subflow.
3. Create the message map by double-clicking the Mapping node to open the **New Message Map** wizard. The wizard detects that the map is being created for the current REST API operation, and preselects the option **Message map with input and output for REST API operation *operation_name***. Click **Finish**.
4. The REST API operation message map is created in the REST API project and opened in the Graphical Data Mapping editor.
5. The input data from the REST API is provided in the map. This is located in the Local Environment, which is automatically added to the map input. Any REST API path or query parameters that are defined for the operation are automatically defined under **Local Environment** > **REST** > **Input** > **Parameters**.

   A **Task** transform is added to the map with its input connected to **Local Environment** > **REST** > **Input**, and enables you to locate the REST API input data that might be required in the mappings. You can remove the Task when you have completed your mappings. The map can be deployed and run with a **Task** transform, with no effect.

   Note: Input form and array type parameters are not supported.
6. The message body of the map input and output is also automatically defined by the REST API Swagger document or OpenAPI 3.0 document.

   The input message body is defined as one of the following types:

   - JSON, with the data type of any request body defined for the operation in the Swagger document or OpenAPI 3.0 document
   - BLOB, when the operation has no request body data

   The ouput message body is defined as one of the following types:

   - JSON, with the data type defined for the HTTP status 200 response. If there is no status 200 response, either the default response or the first response defined in the Swagger document or OpenAPI 3.0 document is used.
   - BLOB, when the operation has no response body data.

   The Data type of the JSON input or output is defined in the following way:

   - If the REST API operation request or response is defined as a model type (JSON reference to a type in the Swagger document or OpenAPI 3.0 document definitions) it has a type name as the model

name; for example, `Pet` or `Department`. If it is an array of a type defined in the model, the type name is prefixed with `JSONArray_`; for example, `JSONArray_Pet`.

- If the API operation request body is defined inline by a Swagger schema statement or OpenAPI 3.0 schema statement, its type name is formed as:

```
<Json type>_<operation name>_body
```

- If the API operation response body is defined inline by a Swagger schema statement or OpenAPI 3.0 schema statement, its type name is formed as:

```
<Json type>_<operation name>_<http status code>
```

For example:

```
string_getSurname_200
```

or

```
object_getItem_200
```

- If the request or response body is defined as an inline array, the type name is prefixed with `JSONArray_`; for example, `JSONArray_string_getAllSurnames_200`

7. Select and delete the output message assembly that was created in the map.
8. Click the **Add an output object** button in the editor to add the required map output, and select the required message format of the input to the intermediary processing (format *X* in the example used earlier in this topic).

   The message format can be any of the supported formats described in "Input and output messages to a message map" on page 1276.

   Note: If the format is defined in a JSON schema model, the JSON schema file must be placed in the REST API project container.
9. Add a second Mapping node as described in Steps #unique_1774/unique_1774_Connect_42_mapadd and #unique_1774/unique_1774_Connect_42_mapwiz.
10. Select and delete the input message assembly that was created in the map.
11. Click the **Add an input object** button in the editor to add the required map input, and select the required message format for the result of the intermediary processing (format *Y* in the example used earlier in this topic).

    The message format can be any of the supported formats described in "Input and output messages to a message map" on page 1276.

    Note: If the format is defined in a JSON schema model, the JSON schema file must be placed in the REST API project container.
12. Complete the required transformation logic in the message maps to implement the REST API operation, using any of the "Transform types in the Graphical Data Mapping editor" on page 1282. For more information, see "Editing message maps" on page 1394.

    In addition to transforming the message data, you can interact with resources as described in the following topics:

    - "Mapping database content" on page 1523
    - "Using Java in a message map" on page 1370
    - "Accessing user-defined properties from a Mapping node" on page 1479

What to do next

Deploy the REST API with the message map and exercise the operation to verify that the output message is as you expected. For more information, see "Troubleshooting a message map" on page 1587.

Deploying message maps

By default, message map files are deployed in BAR files as a part of an application, integration service, or library that provides an integration solution. You can also deploy a map as an independent resource if you are managing your message flows that way. If you change a message map, you must redeploy your integration solution or independent message flows.

About this task

When an integration solution or message flow containing a Mapping node is deployed, message maps that are referenced in the **Mapping routine** property of the Mapping node are resolved and validated. This preparation occurs at deployment time, rather than when the first message is flowed through the node.

This behavior has the following advantages:

- There is no drop in performance from initializing a message map when the first message is flowed through the node.
- The message map and its dependencies, such as any referenced message models, are resolved and validated during deployment to ensure that the message map runs successfully on first message.
- The message map syntax is validated during deployment to ensure that the message map runs successfully on first message.
- When IBM App Connect Enterprise is restarted, the message map syntax and its dependencies are validated before the message flow can be restored.

To avoid a deployment failure, you must include all the message map dependencies, referenced schemas, ESQL modules, Java classes, and other resources in your BAR file. You must resolve any message map static errors such as an invalid XPath expression. If these requirements are not met, you receive a BIP message that reports the map generation failure.

By default, any map that is in the BAR file but not referenced in the **Mapping routine** property of the Mapping node is not validated on deployment. If the map is used dynamically by a Mapping node, by selecting it in a LocalEnvironment override (`LocalEnvironment.Mapping.MappingRoutine = '{brokerSchemaName}:mapName'`), it is validated and prepared when it is first used.

You can configure an integration node or server to validate and prepare all message maps on deployment, by setting the **generateAllMapsOnDeploy** property to `true` in the `server.conf.yaml` configuration file or on the **mqsichangeproperties** command. For example, to set the property for an integration node, you can use the following command:

```
mqsichangeproperties --integration-node integrationNodeName --property generateAllMapsOnDeploy
 --value true
```

To set the property for an integration server (either an independent integration server or a server that is managed by an integration node), you can set it in the `server.conf.yaml` configuration file:

```
generateAllMapsOnDeploy: true
```

The integration server must then be restarted for this change to take effect.

If you set the **generateAllMapsOnDeploy** property to `true`, ensure that all `.map` files in your application, library, and integration project are intended for use by a Mapping node that is in the BAR file. If any other `.map` files are found in the application, library, or integration project, they might cause deployment to fail.

Procedure

When you deploy message maps, the behavior of IBM App Connect Enterprise is as follows:

- Behavior when you deploy or redeploy a BAR file:
  a) All message maps are validated to ensure that all the map dependencies can be resolved at run time. This validation step checks that the referenced message models such as XML schema files, DFDL schema files, and message set files, and the referenced submaps can be resolved.
  b) The message maps and their dependencies are generated to an executable form. This step also checks that the contents of the map and submaps are valid, and that they have no errors such as an invalid XPath expression.
  c) If the message maps and their dependencies are valid and can be successfully generated, they are persisted in both the deployed and generated forms to the configuration store. Otherwise, the deployment is stopped and you receive a BIP message that reports the map generation failure.
- Behavior after deployment:
  a) Background processing is started to compile the generated message maps to Java byte code so that they can benefit from JIT optimization. The Java byte code for each map is persisted on completion of the compilation.
- Behavior when the first message flows runs after deployment or redeployment:
  a) If the background compilation processing is complete, the Mapping node runs the java byte code and the JIT optimization starts.
  b) If the background compilation processing is not complete, the Mapping node runs the map initially in the generated form until the compilation is complete. Then, JIT optimization starts.
- Behavior for any subsequent messages that flow after deployment or redeployment:
  a) The prepared message map runs. If the background processing is completed, the JIT optimization starts.
- Behavior when you restart IBM App Connect Enterprise, that is, when you restart an integration node, an integration server, an integration solution, or a message flow:
  a) If the compiled Java byte code for the map is available, it is loaded and the Mapping node runs this code as soon as the first message is processed. Then, JIT optimization starts.
  b) If the generated form for the map is available and loaded, background processing is started to compile the generated map to Java byte code so that they can benefit from JIT optimization. The Java byte code for each map is persisted on completion of the compilation.
  c) If neither the generated or the compiled map code are available, the map is processed in the same way as when you deploy or redeploy a BAR file.

Troubleshooting a message map

Diagnose and solve problems that you encounter when you use message maps.

Procedure

- In the IBM App Connect Enterprise Toolkit, you can use the Flow exerciser to check the input message that you pass to a map. You can also check the output message of a map after the transformation logic is applied.

  The Flow exerciser is a tool that is available from the flow editor and the integration service editor. You can use the Flow exerciser to perform the following set of tasks:

  1. Deploy the resource to an integration server and set the integration server to recording mode.
  2. Create and send an input message or previously saved recorded message to the input node of the message flow.

     Note: You can also send messages to the message flow by using an external client.
  3. Highlight the message path on the message flow and any subflows that are associated with the message flow.
  4. Display the content of the logical message tree for a message that passed through a connection in the message flow.
  5. Save the content of the logical message tree as a recorded message that you can send to the message flow later.

  For more information, see Testing your message flow by using the Flow exerciser.
- In the development environment, you have different ways to access an error or warning that occurs in a map.

  a) In the **Problems** view, you can double-click a problem or warning. Then, in most cases, the Graphical Data Mapping editor opens and navigates to the transform with the error or warning.

  Note: Sometimes the error is nested deep inside other mappings so it is hard to locate. Double-clicking the error that shows in the **Problems** view takes you to the transform with the error quickly.

  b) In the Graphical Data Mapping editor, you can see an error or a warning as an icon (or marker) associated with the transform where the problem or warning occurs in your map. You can hover over the icon to show the message.

  When an error or warning is present in a nested mapping, the Graphical Data Mapping editor shows the icon on the parent transform.
- During deployment, you cannot add a message map with an **Error** marker to a BAR file until the errors are resolved.

  For more information, see “Deploying message maps” on page 1585.
- In the run time environment, you can use a user trace to troubleshoot transformation logic that you build in your message map. The Mapping node reports the running of a message map as detailed user trace events.

  The user trace events report the entry and completion of each transform in a map, and the setting of values in the map output.

  The collection and review of IBM App Connect Enterprise user trace enables you to troubleshoot transformation logic that you build in your message maps.

  The following topics describe how to diagnose problems by using user trace:

  – Controlling user trace
  – User trace

Using or converting legacy resources into message maps

If you want to deploy legacy message maps (`.msgmap` format) in IBM App Connect Enterprise, you must first convert them into message maps in `.map` format, which the IBM App Connect Enterprise Mapping node can consume.

About this task

Legacy message maps cannot be deployed in IBM App Connect Enterprise until they have been converted.

A legacy message map is a message map that was created as a `.msgmap` file in WebSphere Message Broker Version 7 or earlier.

The functions provided by the following legacy message flow nodes in WebSphere Message Broker Version 7 (and earlier versions) were replaced with the new Mapping node in later versions of WebSphere Message Broker and IBM Integration Bus:

- DataDelete node
- DataInsert node
- DataUpdate node
- Extract node
- Mapping node
- Warehouse node

Procedure

Follow these guidelines when you use legacy message maps and legacy message flow nodes in your integration solutions:

- IBM App Connect Enterprise does not support WebSphere Message Broker Version 7.0 legacy message maps (`.msgmap`). You can import message flows that contain legacy message maps into the IBM App Connect Enterprise Toolkit. However, you can view these message flows in read-only mode only, and you cannot deploy and run them. If you try to deploy a BAR file that contains a legacy message map, you see a BIP2355 error message. Before you run a message flow that contains a legacy message map or a legacy Mapping node, you must convert them by following the instructions in “Using or converting legacy resources into message maps” on page 1588.
- You must change the transformation logic in your integration solution when a legacy message map is invoked from an ESQL CALL statement. For more information, see “Converting a legacy message map that is called from an ESQL statement in a Compute node” on page 1605.

  A message map cannot be called from an ESQL CALL statement.

Changes in behavior in message maps converted from legacy message maps

In IBM App Connect Enterprise, you transform data graphically by using a message map. These maps are managed through the Graphical Data Mapping editor. You define your transformation logic by using XPath 2.0 expressions. You can also call Java methods, ESQL procedures, or complex XPath expressions by using specialized transforms such as the **Custom Java**, **Custom XPath**, or **Custom ESQL** transforms.

Behavior changes during the development phase

Nulls behavior

When you convert a legacy message map (`.msgmap`) that includes handling for nilled elements, check how a message map handles NULL values. For more information, see “Handling nulls in message maps” on page 1368.

- In ESQL, a special NULL value is defined, and is distinct from empty. When you assign NULL to a named element, or set the element from the returned NULL value of a called ESQL function, you delete the element from the tree.

  In a message map, the ESQL NULL produces an empty element, or an empty element with the `xsi:nil` attribute set when the element is defined as nillable in the model. Consequently, in some cases the output of the map might include unexpected empty elements that can cause processing problems, including XML schema validation violations. Such problems typically occur when an ESQL user-defined function that returns ESQL NULL in some conditions is called. To avoid these problems, add a condition to the Custom Transform to prevent it from being invoked if it would return NULL.

Mapping from missing inputs

In ESQL-based legacy message maps, writing a NULL to an output element did not create the element. However, in the converted message map, this action can produce an empty element.

When a legacy message map ran certain transforms, and the input data was missing, the input was set as NULL; therefore, no output element was created. In the converted message map, these cases can produce an empty output element:

- For transforms that are completely self-defined (such as **Move**), and if the input and output elements are defined as optional (`minOccurs=0`), the converted message map handles the missing input element, and no output element is created.
- If the output element is defined as mandatory, the element is created in order to conform to the schema model.
- For transforms that are internally defined to produce some output value even if the input element is missing (such as **Custom ESQL**, **Custom Java**, or **Submap**), the converted message map cannot automatically handle the missing input element. In this situation, you can add a condition such as `fn:exists($input)` to prevent these transforms producing the empty output element.

Some transforms, such as the XPath `fn:concat()` function, require exactly one data value for each parameter. When you are using these types of function, be aware that the legacy `.msgmap` might have provided an empty string for a NULL or missing input; however, the new XPath based `.map` provides an empty sequence, which results in an XPath runtime error because a single data value has not been provided. To solve this problem, you can modify the parameter so that it is wrapped in an XPath coalesce expression, in the following form:

```
(( $input, '' )[1])
```

This coalesce expression causes the parameter to be set to one of the following values:

- The first value from the "`$input`" element, if it exists
- The empty string ", if the "`$input`" element is missing.

Limitations on the conversion of `esql:coalesce` calls

Because there is no XPath equivalent for ESQL NULL, you can convert `esql:coalesce` calls (that have multiple arguments) in a legacy message map only if all of the arguments of the call (except for the final argument) have one of the following three forms:

- A simple source reference; for example, `$source/path/item`.
- An expression without any source reference. For example, the following are valid arguments: 123, `'myString'`, `esql:current-time()`, or `fn:string-length("some-text")`.
- A function (other than `esql:coalesce`), whose arguments also have one of these three forms, and where no more than one argument includes a simple source reference. For example, the following are valid arguments: `fn:current-date()`, `esql:func1(esql:func2($source/path/item))`, or`xs:int(fn:substring($source/path/item, 3, 5))`.

Note: There is no limitation on the conversion of `esql:coalesce` calls that have a single argument, or on the final argument of an `esql:coalesce` call that has multiple arguments.

If the `esql:coalesce` call is in a valid form, and is converted without error, you must review and test the converted transform to ensure that it results in equivalent function.

If you convert a legacy message map that contains an `eqsl:coalesce` call that is not in a valid form, for example, `esql:coalesce(esql:func($source/path/item1,$source/path/item2))`, a **Task** transform is created with details of the error, and you must manually convert the `eqsl:coalesce` call to provide the equivalent function. For more information, see "Managing conversion errors on converted legacy message maps" on page 1598.

Assigning literal values to output elements

Use the Assign transform to set literal values in output elements. The Assign transform uses a string representation, which is assigned to the relevant output element and so must be formatted according to its type. The property value does not need to be in quotation marks, as any quotation marks would be passed as part of the string value. To provide an explicitly typed value, use the `xs:<type>` Cast transform with no input wiring.

Literals in conditional expressions

You could build expressions in the legacy message mapping editor that implied a type cast and used the underlying string value representation.

The Graphical Data Mapping editor uses XPath expression syntax and enforces strict typing. For example, testing a Boolean-typed element for the string literal value **true** would cause a type exception.

You can use the `xs:<type>` functions in your expressions to avoid these incorrect typing issues.

Java methods in conditional expressions

Legacy message maps allowed the use of Java methods with `MbElement` parameters types. The Graphical Data Mapping editor uses standard XPath expression execution and can support only Java methods with standard Java parameter types. For more information, see "Defining a Java conditional expression for a transform" on page 1439.

Complex type text values in condition expression

A legacy message map did not require you to be explicit when accessing mixed content text values from a complex type element in a condition expression.

The Graphical Data Mapping editor is based on standard XPath syntax, and requires the explicit use of **/text()** to signify that the mixed content text value is to be used. As a result, a converted map with a conditional expression that referenced mixed content text values might fail until the path expression is extended to add the missing **/text()**.

Literals in submap calls

The legacy message map editor did not correctly validate the typing of submap inputs. You could edit the normal element path value of a submap input, and instead provide an untyped literal value.

The Graphical Data Mapping editor requires that all submap inputs are wired to an appropriately typed input element.

"For each index" counter variables

Some transformations require the use of the "For each index" counter value. The WebSphere Message Broker Version 7 message map provided the **`msgmap:occurrence`** function to obtain the current loop count. The Graphical Data Mapping editor provides a **`For loop`** counter variable which can be used to provide equivalent function. The name of this variable is fixed format `$<For each primary input element name>-index` can be obtained using content assist `ctrl-space` in the relevant "ForEach transform" **Filter properties expression** panel, or in the content assist in any nested transforms.

Explicit casting of variables in conditional expression

While using conditional expressions like the `If` transform, for complex expressions or when a return value from ESQL function was being compared with a variable, you could add explicit type casting in expressions using `xs:<type>( $<var> )`. An example of an `If` transform for checking the variable of DATE datatype: function:

```
isStartdate() =xs:date($test)
```

Behavior changes during the deployment phase

In IBM App Connect Enterprise, message maps can be deployed only as source. You must provide any of the following resources before deploying your solution:

- For text and binary messages, you must provide the DFDL schema file or the message set that defines your input and output messages.
- For XML messages, you must provide the XML schema files that define your input and output messages.

If your message is modeled in a message set, the message map requires the message set schema (`.xsdzip` file) to be deployed to run your message map. If your existing message set is used for text and binary formats only, you can deploy your message map with only a `.dictionary` file in the BAR file. In this case, you must modify the message set to additionally set the XMLNSC domain support option, so it is added to a BAR file with both a `.dictionary` file and `.xsdzip` file. If this option is not set, a warning is displayed in the **Problems** view, along with a quick fix action.

Behavior changes at run time

- In IBM App Connect Enterprise, the Graphical Data Mapping editor has a dedicated Java based run time. As a result, map execution benefits from full support for XPath 2.0 and Java JIT optimization, which offers increased reliability.
- A message map can be invoked only from a Mapping node.
- A submap can be invoked only from a top level message map, that is, a map consumed by a Mapping node.
- The message map runs in the Java virtual machine of the integration node. You must ensure that the integration node is configured correctly. For more information, see "Setting the JVM heap size" on page 2762.

Timezone handling

If you use a Mapping node that includes XPath datetime, or Assign transforms to set the value of output elements that are defined as `xs:date`, `xs:dateTime`, or `xs:time` and you want the timezone included, you must set the environment variable MQSI_USE_TIMEZONE to any value.

Planning the conversion of a legacy message map
Before you convert a legacy message map, review this section to help you plan the migration.

About this task

If you want to deploy legacy message maps (`.msgmap` format) in IBM App Connect Enterprise, you must first convert them into message maps in `.map` format, which the IBM App Connect Enterprise Mapping node can consume.

Legacy message maps, which have not been modifiable since WebSphere Message Broker Version 7, could be deployed in later versions of WebSphere Message Broker; however, they cannot be deployed in IBM App Connect Enterprise until they have been converted.

Procedure

Complete the following tasks to plan the conversion of a legacy message map to an IBM App Connect Enterprise message map:

1. Verify that all the projects that include resources used by the legacy message map are available in your workspace, and the project dependencies are defined.

   The conversion process needs access to all projects that include resources used by the legacy message map, so that it can convert your map transformations automatically.
2. Identify the legacy message map resources used in your transformations:
   a) Check the input and output structures of the legacy message map.
      - Are you doing transformations that include the local environment tree?
      - Are you doing transformations that include data structures with *xsd:any* elements?

      For more information, see "Converting a legacy message map that includes transformations of the local environment tree or xsd:any elements" on page 1600.
   b) Check your transformation for any of the following types:
      - Transformations that include calls to user-defined ESQL procedures. For more information, see "Converting a legacy message map that includes user-defined ESQL procedures" on page 1601.
      - Transformations that include calls to ESQL mapping functions. For more information, see "Converting a legacy message map that includes ESQL mapping functions" on page 1602.
      - Transformations that include calls to message map functions. For more information, see "Converting a legacy message map that includes calls to message map functions" on page 1604.
      - Transformations that include calls to relational database operations. For more information, see "Converting a legacy message map that includes relational database operations" on page 1605.
      - Transformations that include calls to a message map. For more information, see "Converting a legacy message map that is called from an ESQL statement in a Compute node" on page 1605.

What to do next

1. Define schema models for any *xsd:any* element in your input or output structures. You can also use the **Add user-defined** function to qualify the *xsd:any* element. This approach is recommended when the *xsd:any* element only involves a small number of elements. If you reuse the same structure in multiple maps, it is recommended that you create a schema model.
2. Run the conversion process. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.
3. Review the newly created message map and complete the post-conversion tasks. For more information, see "Managing conversion warnings on converted legacy message maps" on page 1594 and "Managing conversion errors on converted legacy message maps" on page 1598.
4. Update the message flow that includes the legacy message map to include the new Mapping node and reference the newly created message map. For more information, see "Replacing a legacy Mapping node" on page 1606.
5. Deploy and test your message flow. For more information, see Deployment rules and guidelines.

Converting a message map from a `.msgmap` file to a `.map` file

You must convert legacy message maps (`.msgmap` format) to graphical data maps (`.map` format) before you can deploy or modify them.

Before you begin

Before you convert a legacy message map, complete the following steps:

1. Import your resources from WebSphere Message Broker. For more information, see Importing resources from previous versions. Alternatively, you can migrate the Toolkit development resources. For more information, see Migrating development resources.
2. Verify that all the projects that include resources used by the legacy message map are available in your workspace, and the project dependencies are defined.

3. Review "Changes in behavior in message maps converted from legacy message maps" on page 1588 and "Considerations for mapping messages modeled in message sets" on page 1274.

About this task

In IBM App Connect Enterprise, if you want to deploy or modify a legacy message map (`.msgmap` format), you must first convert it to a graphical data map (`.map` format). A graphical data map is known as a message map in IBM App Connect Enterprise.

Note: The conversion process is not reversible. However, you can run the conversion of a legacy message map multiple times. You must rename the converted legacy message map by removing *_backup* from the converted map name.

To use a converted legacy message map in your message flows, you must replace legacy Mapping nodes with new Mapping nodes. For more information, see "Replacing a legacy Mapping node" on page 1606.

Procedure

To convert a legacy message map to a message map by using the IBM App Connect Enterprise Toolkit, complete the following steps:

1. Start the conversion process: In the **Application Development** view, right-click the message map that you want to convert, and click **Convert Message Map from .msgmap to .map**.

   To covert multiple legacy message maps, right-click a folder, project, application, or library that contains one or more maps, and click **Convert Message Map from .msgmap to .map**.

   - Your converted message map is created, and is displayed in the **Application Development** view.
   - Your legacy message map is renamed *MessageMapName*`.msgmap_backup`, and is displayed in the **Application Development** view.
2. Open the converted map in the Graphical Data Mapping editor: In the **Application Development** view, double-click your new message map.

   The message map opens in the Graphical Data Mapping editor.
3. Review the transformation logic that was created by the conversion process to ensure that it produces the correct output for your application:
   a) Review and replace each **Task** transform. For more information, see "Managing conversion errors on converted legacy message maps" on page 1598.

      If your legacy message map contains complex mapping structures that the conversion process was not able to recreate, your message map includes **Task** transforms to assist you in manually recreating those structures.

      Task transforms are listed in the **Problems** view.

   b) Review all the conversion annotation icons () that are associated to transforms in your map. For more information, see "Managing conversion warnings on converted legacy message maps" on page 1594.

      Conversion annotation icons are displayed on the lower left of the transform in the Graphical Data Mapping editor.

      You can accept or reject all the transforms in a converted map in a single click.

Results

Your legacy message map is converted to a message map that can be deployed, and that can be modified by using the Graphical Data Mapping editor.

What to do next

Modify each message flow that references the legacy message map so that new Mapping nodes reference your new message map. For more information, see “Replacing a legacy Mapping node” on page 1606.

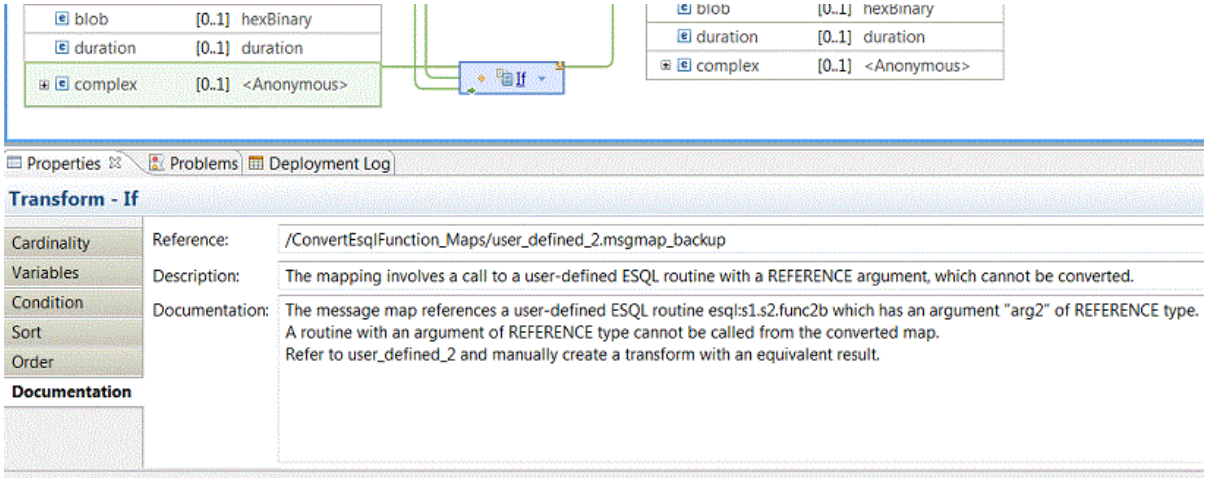
Managing conversion warnings on converted legacy message maps
You can accept or reject conversion actions suggested by the map conversion process when you transform a legacy message map into a message map.

About this task

If you convert a legacy message map (`.msgmap` file) to a message map (`.map` file), and your message map contains complex mapping structures that the conversion process is able to re-create, your new map might include warnings on one or more of the transforms that are created by the conversion process.

These warnings are displayed in the Graphical Data Mapping editor as conversion annotation icons ( ).

Note: The conversion process creates one warning per transform that needs to be reviewed. You must check all the warnings present on one or more of the transforms in your converted map because the output of the converted transform might be different to the behavior in your original message map.



If a transform contains a conversion warning, it will have a small green arrow in the bottom-left of the transform.

Procedure

In the Graphical Data Mapping editor, open your converted map, and complete the following actions to resolve the conversion warning:

1. Select a transform with a warning.
2. Review the transform properties in the **Properties** view to ensure that the transformation logic is correct.

3. Accept or reject the proposed transform. Select one of the following actions:
   a) Select **Accept Converted Transform** to remove the conversion warning and accept the transformation logic implemented by the conversion process.
   b) Select **Reject Converted Transform** to remove the warning. The conversion process replaces the converted transform with a **Task** transform. Then, manually re-create the transformation logic.
      i) Select the **Task** transform.
      ii) In the **Task** transform **Properties** view, check out the information provided in the **Documentation** tab.
      iii) Edit the **Condition** property of the transform and manually define the transformation logic.

      In the **Documentation** tab of the transform, the map conversion process includes the mapping structure from your legacy message map.
   c) Select **Accept All Converted Transforms** to remove all the conversion warnings in your map and accept all the transformation logic implemented by the conversion process.
   d) Select **Reject All Converted Transforms** to remove all the conversion warnings. The conversion process replaces each transform with a **Task** transform. Then, manually re-create the transformation logic.

What to do next

Check the following conversion warning descriptions and the actions you can take to resolve them:

Conversion warning description	Action to resolve it
The Java method call {0} cannot be converted. Before conversion, a Java method should be have been `java:[package name].[class name].[method name]([arguments])` Refer to {1} and create a Custom Java Transformation.	The mapping cannot be converted due to a syntax error in the Java method call {0}. To automatically convert the Java method, provide a "java:[package name].[class name].[method name]([arguments])". Alternatively, define a Custom Java Transform and implement your Java code.
The message map references the function {0}. The numbers of inputs provided by the message map and expected by the function definition are {1} and {2}, respectively. The extra input is:{3} Refer to {4} to resolve the discrepancy.	Review the input connections to the transform and remove the one that is not needed. For more information, see “Converting a legacy message map that includes ESQL mapping functions” on page 1602.
The message map references the function {0}. The numbers of inputs provided by the message map and expected by the function definition are {1} and {2}, respectively. Refer to {3} and provide values for all inputs.	Review your input connections and add the missing ones. The transform is missing at least one input connection For more information, see “Converting a legacy message map that includes ESQL mapping functions” on page 1602.
The following submap input in the message map was an expression not corresponding to any mapping input. Connect the input to a mapping input which will provide the value of the expression. Refer to {0} for the original mapping.	Connect the missing inputs to the submap.

Conversion warning description	Action to resolve it
The signature for Java method {0} in class {1} is not found in the workspace within the referenced projects and the Java build path. Refer to {2} for the original mapping and make sure the Java method is accessible from the map.	Define a project reference from the project containing the message map to the Java project that contains the Java class and method.
Review the definition being used as the cast for {0}, since it is defined in the following schemas. {1}	Check that the schema used to cast an `xsd:any` element or attribute is the correct one.
The connection to input {0} was converted from expression {1}. There is more than one element that can be identified as {1}. Review the input connection, and move the connection to another {0} if appropriate.	The transform was converted from a mapping with an ambiguous input.
The transformation code was converted from {0}. It is advised to review to ensure equivalent transformation outcome. Refer to {1} for the original function usage.	Review the transformation logic.
The transform was converted from a "for" statement, and produces one output for each instance of the input {0}. However, the output of the transform {1}, by definition, may expect only one occurrence. User review is advised to ensure the output is as expected. Refer to {2} for the original mapping.	Review the transformation logic.
The function {0} was converted from {1}. It is advised to review to ensure equivalent transformation outcome. Refer to {2} for the original function usage.	Review the transformation logic.
The transform was created by converting a non-trivial mapping in an older version of message map. User review is advised to ensure equivalent transformation outcome. Refer to {0} for the original mapping.	Review the transformation logic.
The transform was converted from a call to a user-defined ESQL function. All inputs of this transform are optional, if the user-defined ESQL function returns NULL when none of the inputs are present, an unexpected empty output element could be produced. To avoid an empty output element, consider setting a condition on the transform as shown below so that the transform will only be performed when at least one input exists: t{0} Refer to {1} for the original mapping.	A condition might be needed on the transform.

Conversion warning description	Action to resolve it
The message map references a user-defined ESQL routine {0} which has an argument {1} of REFERENCE type. A routine with an argument of REFERENCE type can be called from the converted map only if the following statements are true: • The reference variable is used to access only the input element or descendants of the input element. • The input element and any descendants that are accessed by using the reference variable are defined in the input message model. Refer to {2}, review and test the transform to ensure an equivalent result.	Check that the converted map gives the correct result.
The transform was converted from an XPath expression involving an XML type cast. Before conversion, XPath functions were implemented in the runtime using equivalent ESQL functions. Some of these ESQL functions have more lax value typing than defined in the XPath specification of the function. The Graphical Data Mapper provides conforming XPath functions. As a result, the transform may fail at runtime with invalid value for type casting issues. In particular, the ESQL equivalent function may have provide a default value when the parameter value was empty. To resolve these type of issues, add conditions to detect and prevent the values that are invalid for the XPath function parameters being passed to the function. For example, the following condition will make the transform executed only when all inputs exist: \t{0} Refer to {1} for the original mapping.	A condition might be needed on the transform. For more information, see “Converting a legacy message map that includes ESQL mapping functions” on page 1602.
The XPath expression was converted from an expression involving a call to the msgmap:exact-type function. It is advised to review to ensure equivalent transformation outcome. Refer to {0} for the original function usage.	Review the XPath expression. For more information, see “Converting a legacy message map that includes calls to message map functions” on page 1604.
The transform involves the following input element which involves a text content: Determine if the element or its text content is the intended mapping input or output. If necessary, modify the corresponding connection. Refer to {0} for the original mapping.	Review the XPath expression. For more information, see “Changes in behavior in message maps converted from legacy message maps” on page 1588.
The transform involves the following output element which involves a text content: Determine if the element or its text content is the intended mapping input or output. If necessary, modify the corresponding connection. Refer to {0} for the original mapping.	Review the XPath expression. For more information, see “Changes in behavior in message maps converted from legacy message maps” on page 1588.

Conversion warning description	Action to resolve it
The XPath function call was converted from {0}. Ensure the argument number, order and type matches the supported function specification. Refer to {1} for the original function usage.	Review the XPath expression against the XPath specification.

Managing conversion errors on converted legacy message maps
The conversion process creates a **Task** transform with an error when it cannot automatically convert a legacy message map transformation. You can use any of the transforms in the Graphical Data Mapping editor to reconstruct an equivalent transformation. Typically, you might use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic.

About this task

These errors are displayed in the Graphical Data Mapping editor as **Task** transforms marked with an error.

You can see the error description associated with the conversion in the **Problems** view or by hovering over the red exclamation symbol associated to a **Task** transform.

Procedure

In the Graphical Data Mapping editor, open your converted map, and complete the following actions to resolve the error:

1. Select a **Task** transform.

   The transform properties are displayed in the **Properties** view.
2. In the **Properties** view, click the **Documentation** tab to review details about the mapping structure that was not re-created by the conversion process.

   In the **Documentation** tab of the transform, the map conversion process includes the mapping structure from your legacy message map. For example, you can find the following description for conversion of unsupported legacy transforms:

   ```
   The expression {0} used for {1} cannot be automatically converted into a supported transform.
    \n\
   Refer to {2} and manually create a transform with an equivalent expression.
   ```
3. Change the **Task** transform to a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform.

   Click the arrow in the transform box, and select a transform from the list of available ones. For more information, see “Editing message maps” on page 1394.
4. Configure the transform properties to manually re-create the transformation logic.

What to do next

Check the following conversion error descriptions and the actions you can take to resolve them:

Conversion error descriptions	Action to resolve it
The statement {0} cannot be automatically converted into a supported transform. Refer to {1} and manually create mappings to perform the corresponding transformation.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic.

Conversion error descriptions	Action to resolve it
The expression {0} cannot be automatically converted into a supported transform. Refer to {1} and manually create a transform with an equivalent expression.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic. If the expression is `esql:coalesce`, see "Limitations on the conversion of esql:coalesce calls" on page 1589.
The expression {0} used for {1} cannot be automatically converted into a supported transform. Refer to {2} and manually create a transform with an equivalent expression.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic.
The XPath function call {0} for {1} is not supported. Refer to {2} and manually create a transform with an equivalent expression.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic.
The message map contains an erroneous expression. An attempt is made to convert the expression. Refer to {0} and review the converted expression.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic.
The message map references a user-defined ESQL routine {0} which has an argument {1} of REFERENCE type. A routine with an argument of REFERENCE type cannot be called from the converted map. Refer to {2} and manually create a transform with an equivalent result.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic. For more information, see "Converting a legacy message map that includes user-defined ESQL procedures" on page 1601.
The message map references a user-defined ESQL routine {0} which has an INOUT argument {1}. When a routine with an INOUT argument is called from the converted map, the argument must behave like an IN argument. Refer to {2} and the ESQL routine.	Check the **Documentation** tab of the **Task** transform to get details of the original expression. Use a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform to re-create the transformation logic. For more information, see "Converting a legacy message map that includes user-defined ESQL procedures" on page 1601.

Converting a legacy message map that includes transformations of the local environment tree or xsd:any elements

Before you convert a legacy message map that includes transformations of the local environment tree or *xsd:any* elements, you must provide the XML schema of the input and output data structure in a library. The library with the schemas must be visible by the project hosting the imported legacy message map.

About this task

In a message map, the Variables folder in the local environment tree is represented by *xsd:any*. In IBM App Connect Enterprise, you must qualify the Variables folder to provide the elements for your map. For more information, see "Mapping data in the local environment tree" on page 1464.

When you convert a legacy message map, you can encounter any of the following conversion behaviors:

- You have an *xsd:any* element and the schema model associated to the message set that you use to qualify it in your map. The conversion process casts the *xsd:any* element to the schema model automatically.
- You have an *xsd:any* element and no schema model describing its structure. The conversion process fails the first time. You must define the model and run again the conversion process.
- You have a legacy message map where you edit the path expression in your map to define the element that must be read to qualify the *xsd:any* element. The conversion process fails the first time. You must define the schema model and run again the conversion process.

If you only need to qualify a small number of elements, then, you might want to consider using the **Add user-defined** function instead of defining a schema model. For more information, see "Defining user-defined elements" on page 1408.

Procedure

In the **Application Development** view, complete the following steps to convert a legacy message map that includes transformations of the local environment tree and of *xsd:any* elements:

1. If you use the local environment tree in your legacy message map transformations, create an XML schema model in a library project. The model must define the Variables folder data structure.
2. If your input message or your output message include *xsd:any* elements that you use as part of your transformations, define the XML schema model for each one.
3. Start the conversion process: Right-click the message map that you want to convert, and click **Convert Message Map from .msgmap to .map**.
4. Open the converted map in the Graphical Data Mapping editor: Double-click your new message map.
5. Review the conversion errors and build the list of unresolved elements. For more information, see "Managing conversion errors on converted legacy message maps" on page 1598.
6. Create XML schema models for each unresolved data structure in a library that is referenced by the project hosting your converted map.

   For more information, see "Mapping data in the local environment tree" on page 1464.
7. Delete the converted legacy message map.
8. Rename the converted legacy message map by removing *_backup*.
9. Rerun the conversion process: Right-click the message map that you want to convert, and click **Convert Message Map from .msgmap to .map**.

   The conversion process casts automatically your *xsd:any* elements. Any related errors to unresolved elements disappear.

What to do next

Continue converting your legacy message map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

Converting a legacy message map that includes user-defined ESQL procedures
When you convert a legacy message map that includes ESQL procedures, the conversion process converts each ESQL procedure to an equivalent **Custom ESQL** transform that invokes the ESQL. A **Task** transform is added to your converted map when an ESQL procedure does not fulfill the requirements to be called from a map on a Mapping node.

Procedure

Check the conversion process behavior when you convert a legacy message map to a message map that includes ESQL procedures:

1. Check that the conversion meets the requirements for ESQL modules that are called from a graphical data map; see "Requirements for ESQL modules that are called from a graphical data map" on page 1309.
2. Each converted ESQL procedure is deployed as source. If you are not using IBM App Connect Enterprise application and library projects to store your ESQL procedures, check that the ESQL procedures have unique names because they are deployed independently to the same integration server.
3. Check whether the legacy message map includes ESQL procedure that uses the ESQL REFERENCE data type.

   By default, the conversion process converts ESQL procedures as follows:

   - An ESQL procedure that does not use the ESQL REFERENCE data type is converted to **Custom ESQL** transform. For more information, see "Custom ESQL" on page 1308.
   - An ESQL procedure that uses the ESQL REFERENCE data type is converted to **Task** transform. You must then replace the **Task** transform to complete the map conversion. You can replace the **Task** transform with a **Custom XPath** transform that provides equivalent function. You can also use a **Custom Java** transform or a **Custom ESQL** transform.

   If you have confirmed the requirements that are associated with the ESQL REFERENCE data type are met (see "Requirements for ESQL modules that are called from a graphical data map" on page 1309), and you want an ESQL procedure that uses the ESQL REFERENCE data type to be converted to a **Custom ESQL** transform instead of a **Task** transform, set a preference by completing the following steps:

   a. From the IBM App Connect Enterprise Toolkit menu, click **Windows** > **Preferences** > **Integration Development** > **Convert Message Map from .msgmap to .map**
   b. Select the check box for **Allow the conversion of user-defined ESQL procedures that include REFERENCE parameters**.
4. Check whether the ESQL procedure has an INOUT argument.

   The conversion process converts an ESQL procedure that has an INOUT argument to a **Custom ESQL** transform where the INOUT argument is converted as an IN argument. You can replace the **Custom ESQL** transform with a **Custom XPath** transform, or a **Custom Java** transform when the default conversion transform is not valid.
5. Consider whether the ESQL procedure can return a NULL value.

   In a legacy message map that is executed in generated ESQL, the output target is not created if the ESQL procedure returns a NULL value. In the converted `.map` file, the target output element is created with an empty value if the ESQL procedure returns a NULL value. It might be necessary to add an XPath condition expression to the **Custom ESQL** transform in the converted `.map` file to prevent the ESQL being called in situations where it would return an ESQL NULL value. For more information, see "Nulls behavior" on page 1588.

What to do next

Continue converting your legacy message map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

Converting a legacy message map that includes ESQL mapping functions
When you convert a legacy message map that includes ESQL mapping functions, the conversion process converts some ESQL functions to equivalent XPath 2.0 functions (`fn:functionName`), or to cast type functions (`xs:type`). A **Task** transform is added to your converted map when there is no automatic conversion for an ESQL function.

Before you begin

If your legacy message map includes `esql:coalesce` calls, check that the format of the calls can be converted; see "Limitations on the conversion of esql:coalesce calls" on page 1589.

Procedure

Check the conversion process behavior when you convert a legacy message map to a message map that includes ESQL mapping functions:

1. When a legacy message map includes calls to predefined ESQL mapping functions, each ESQL function call is converted to an XPath expression, cast type function, or to a **Custom XPath** transform in the converted map. For each expression, `xs:type` function, or **Custom XPath** transform in the converted map, complete the following steps:
   a) Check that the expression, `xs:type` function, or **Custom XPath** transform re-creates the required behavior.

      If your ESQL mapping function has optional input parameters, you must implement conditions to handle this situation. By default, the conversion process assumes that all input parameters are mandatory.

   b) For each expression, `xs:type` function, or transform, check that the correct number of inputs is connected.

      In earlier releases of WebSphere Message Broker Version 8, the number of inputs wired to a transform and required to implement a transformation in a legacy message map was not enforced. When the Graphical Data Mapping editor converts a transform that includes an ESQL mapping function, it creates an XPath function that conforms to the XPath 2.0 specification, and wires the input elements to the transform as defined in the legacy message map. As a result, a converted map might have more inputs than the XPath expression requires, or less inputs than the ones required to perform the calculation. Consequently, the converted map will fail to run when you deploy it.
2. If there is no XPath equivalent of an ESQL mapping function, the function is replaced with a **Task** transform in your converted map. You must replace each of these **Task** transforms with a **Custom**

XPath transform, a **Custom Java** transform, or a **Custom ESQL** transform that re-creates the required behavior.

a) Check the **Documentation** properties of the transform in the converted map for more information on how the ESQL function was implemented in your legacy message map.

The following ESQL mapping functions that you can use in a legacy message map have no XPath equivalent in message maps:

- Certain mathematical functions:
  - ACOS
  - ASIN
  - ATAN
  - ATAN2
  - BITAND
  - BITNOT
  - BITOR
  - BITXOR
  - COS
  - COSH
  - COT
  - DEGREES
  - EXP
  - LN
  - LOG
  - LOG10
  - MOD
  - POWER
  - RADIANS
  - RAND
  - SIGN
  - SIN
  - SINH
  - SQRT
  - TAN
  - TANH
- Decimal function:
  - TRUNCATE
- Certain String functions:
  - LTRIM
  - RTRIM
  - TRIM-LEADING
  - TRIM-TRAILING
  - REPLICATE
  - SPACE
  - TRIM-BOTH(Singleton FROM Source)

The simple form TRIM-BOTH (Source) is converted.

- Certain field functions:
  - ABITSTREAM
  - BITSTREAM
  - SAMEFIELD
- Certain date time functions:
  - TIMESTAMP
  - CURRENT-GMTDATE
  - CURRENT-GMTTIME
  - CURRENT-GMTTIMESTAMP
- All INTERVAL- functions
- The ESQL LIKE function
- The ESQL FOLLOWING form of the ESQL POSITION function
- All SQL functions
- The UUIDASCHAR and UUIDASBLOB functions

What to do next

Continue converting your legacy message map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

Converting a legacy message map that includes calls to message map functions
The conversion process converts a number of supported calls to predefined legacy message map functions (`msgmap:functionName`) to an XPath expression, or a **Custom XPath** transform.

About this task

If there is no XPath equivalent of a message map function, the conversion process creates a **Task** transform instead.

The following legacy message map functions have no XPath equivalent:

- msgmap:element-from-bitstream
- msgmap:cdata-element
- msgmap:db-path

Procedure

Complete the following steps to convert a transformation that includes a call to a message map function that is not converted automatically:

1. Select a **Task** transform.
2. In the **Task** transform **Properties** view, check out the information provided in the **Documentation** tab.
3. Replace the **Task** transforms with any of the available transforms. Consider using a **Custom XPath** transform, a **Custom Java** transform, or a **Custom ESQL** transform.
4. Manually re-create the equivalent transformation logic of the message map function.

What to do next

Continue converting your legacy message map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

Converting a legacy message map that includes relational database operations
You must manually convert a transformation that includes relational database operations.

About this task

A legacy Mapping node connects to a database via ODBC. A database configuration file describes the ODBC configuration to your database. You configure the database information in your legacy message map by setting the **Data Source** property in the map.

In IBM App Connect Enterprise, the Mapping node connects to databases via JDBC type 4 connections. You configure the database connection details in a JDBC Providers policy.

Database transforms in a message map use support for JDBC connections that are built into the IBM App Connect Enterprise run time.

For information about constructing database transforms and configuring your JDBC database connection, see "Mapping database content" on page 1523.

When you convert a legacy message map that includes database operations, the conversion process creates a **Task** transform for each database operation in the legacy message map. You have to manually replace that **Task** transform.

Procedure

To recreate the transformation logic of a transformation that includes a relational database operation, you can use any of the following transforms when replacing the **Task** transform in the new message map:

- "Select" on page 1338
- "Update" on page 1342
- "Delete" on page 1319
- "Database Routine" on page 1318
- "Insert" on page 1332

What to do next

Continue converting your legacy message map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

Converting a legacy message map that is called from an ESQL statement in a Compute node
To convert a legacy message map that is called from an ESQL CALL statement in a Compute node, you must implement your ESQL logic in a Mapping node, and call the converted legacy message map through a **Submap** transform.

Before you begin

Review "Changes in behavior in message maps converted from legacy message maps" on page 1588 and "Considerations for mapping messages modeled in message sets" on page 1274.

About this task

In IBM App Connect Enterprise, a graphical data map cannot be called from ESQL statements in a Compute node. You cannot modify or deploy a legacy message map in a BAR file.

Before you can modify or deploy a legacy message map (`.msgmap` format), you must convert it to a message map (`.map` format) and modify your message flow logic.

Procedure

To convert a legacy message map that is called by an ESQL CALL statement in a Compute node, complete the following steps:

- Replace the Compute node and the legacy message map with a new message map and Mapping node:
  a) Convert the legacy message map (`.msgmap`) to a new message map (`.map`), as described in "Converting a message map from a .msgmap file to a .map file" on page 1592.
  b) Replace each Compute node that includes a call to the legacy message map with a Mapping node.
  c) Create a new message map for each Mapping node.

     In this message map, complete the following steps:

     a. Create a message model that defines the overall input and output data structure to the Mapping node.
     b. Define the transforms that implement equivalent logic to the ESQL routine.
     c. Replace the CALL statement with a **Submap** transform. For more information, see "Submap" on page 1339.
     d. Configure the converted legacy message map as the mapping routine of the **Submap** transform.
- Replace the called message map with an ESQL function that provides equivalent logic.

What to do next

Deploy and test your message flow. For more information, see Deployment rules and guidelines.

Replacing a legacy Mapping node
A legacy Mapping node (created in WebSphere Message Broker Version 7), cannot reference a graphical data map (`.map` file). If you want to reference a graphical data map in IBM App Connect Enterprise, you must replace the legacy Mapping node with a new Mapping node, which is supported in IBM App Connect Enterprise.

Before you begin

Before you replace a Mapping node, you must complete the following task:

- Import your resources from IBM App Connect Enterprise. For more information, see Importing resources from previous versions.

You might also need to complete the following task:

- Convert a message map to a graphical data map. For more information, see "Converting a message map from a .msgmap file to a .map file" on page 1592.

About this task

If you import your integration solutions from WebSphere Message Broker Version 7, you must convert your legacy message maps and Mapping nodes before you can deploy the message flows that use them.

If you want to reference a graphical data map (`.map` file) in a Mapping node, you must use a Mapping node that was created in WebSphere Message Broker Version 8 (or later) or in IBM App Connect Enterprise.

To replace a legacy Mapping node by using the IBM App Connect Enterprise Toolkit, complete the following steps:

Procedure

1. In the **Application Development** view, double-click a message flow that contains one or more legacy Mapping nodes.

   The message flow opens in the Message Flow editor.
2. In the Message Flow editor, identify a legacy Mapping node that you want to replace.

3. In the Palette, expand the **Transformation** section, and then drag a new Mapping node from the Palette to the canvas of the Message Flow Editor.

   A new Mapping node is added to your message flow, and is assigned a default name.

   If you rename the node, the name that you choose must be unique in the message flow.

   If you do not change the default name at this time, you can change it later, as described in Renaming a message flow node.
4. Select your new Mapping node.

   The node properties are displayed in the Properties view.
5. In the Properties view, a default value is entered in the *Mapping routine* property, and must be replaced by choosing one of the following actions:
   - To reference an existing graphical data map, click **Browse...** to locate it, or specify your `.map` file in the format `{`*`BrokerSchemaName`*`}:`*`MapName`*. `{default}` indicates that no Broker schema is used by the graphical data map. For more information, see "Referencing an existing message map from a Mapping node" on page 1537.
   - To create a new graphical data map, double-click the Mapping node, or right-click the Mapping node and click **Open Map**. For more information, see "Creating a message map from a Mapping node" on page 1385.
6. Move the existing connections from your legacy Mapping node to your new Mapping node. For more information, see Connecting message flow nodes.
7. Select your legacy Mapping node and press the delete key (**del**) to remove it from your message flow. For more information, see Removing a message flow node.
8. Repeat steps "3" on page 1607 through "7" on page 1607 for each legacy Mapping node that you want to replace in your message flow.

Results

You have removed your legacy Mapping nodes and replaced them with new Mapping nodes.

What to do next

Deploy and test your message flow. For more information, see Deployment rules and guidelines.

Developing ESQL

Customize processing implemented by the Compute, Database, DatabaseInput, and Filter nodes in your message flows by coding ESQL.

About this task

You must create, for each node, an ESQL module in which you code the ESQL statements and functions to tailor the behavior of the node. You can access message content, or database content, or both, to achieve the results that you require. ESQL modules are maintained in ESQL files, managed through the Integration Development perspective.

This section provides the following information:

- "ESQL overview" on page 1608
- "Managing ESQL files" on page 1615
- "Writing ESQL" on page 1629

You can use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is run.

ESQL overview

Extended Structured Query Language (ESQL) is a programming language defined by IBM App Connect Enterprise to define and manipulate data within a message flow.

This section contains introductory information about ESQL.

- For descriptions of ESQL user tasks, see "Writing ESQL" on page 1629.
- For reference information about ESQL, see ESQL reference.

Read the following information before you proceed:

- An overview of message flows in "Message flows overview" on page 483.
- An overview of message trees in "The message tree" on page 500, and the topics within this container, paying special attention to "Logical tree (message assembly)" on page 501.

ESQL is based on Structured Query Language (SQL) which is in common usage with relational databases such as DB2. ESQL extends the constructs of the SQL language to provide support for you to work with message and database content to define the behavior of nodes in a message flow.

The ESQL code that you create to customize nodes within a message flow is defined in an ESQL file, typically named `<message_flow_name>.esql`, which is associated with the integration project. You can use ESQL in the following built-in nodes:

- node
- node
- node
- node

You can also use ESQL to create functions and procedures that you can use in the Mapping node.

To use ESQL correctly and efficiently in your message flows, you must also understand the following concepts:

- Data types
- Variables
- Field references
- Operators
- Statements
- Functions
- Procedures
- Modules

Use the ESQL debugger, which is part of the flow debugger, to debug the code that you write. The debugger steps through ESQL code statement by statement, so that you can view and check the results of every line of code that is run.

ESQL data types overview

A data type defines the characteristics of an item of data, and determines how that data is processed. ESQL supports six data types, listed later in this section. Data that is retrieved from databases, received in a self-defining message, or defined in a message model (using MRM data types), is mapped to one of these basic ESQL types when it is processed in ESQL expressions.

Within an integration node, the fields of a message contain data that has a definite data type. It is also possible to use intermediate variables to help process a message. You must declare all such variables with a data type before use. A variable's data type is fixed; If you try to assign values of a different type you get either an implicit cast or an exception. Message fields do not have a fixed data type, and you can assign values of a different type. The field adopts the new value and type.

It is not always possible to predict the data type that results from evaluating an expression. This is because expressions are compiled without reference to any kind of message schema, and so some type errors are not caught until run time.

ESQL defines the following categories of data. Each category contains one or more data types.

- Boolean
- Datetime
- Null
- Numeric
- Reference
- String

ESQL variables overview
An ESQL variable is a data field that is used to help process a message.

You must declare a variable and state its type before you can use it. The data type of a variable is fixed; if you code ESQL that assigns a value of a different type, either an implicit cast to the data type of the target is implemented or an exception is raised (if the implicit cast is not supported).

To define a variable and give it a name, use the DECLARE statement.

The names of ESQL variables are case-sensitive; therefore, make sure that you use the correct case in all places. The simplest way to guarantee that you are using the correct case is always to define variables using uppercase names.

The IBM App Connect Enterprise Toolkit marks variables that have not been defined. Remove all these warnings before deploying a message flow.

You can assign an initial value to the variable on the DECLARE statement. If an initial value is not specified, scalar variables are initialized with the special value NULL, and ROW variables are initialized to an empty state. Then, you can change the value of a variable by using the SET statement.

ESQL Variables declared at Module level 'belong' to a single node. However, variables declared at the Schema level are also given to each node that references that schema. So although variables at schema level are only declared once, each ESQL node has its own copy, which is not shared with any other node (unless the variable is marked SHARED).

Three types of built-in node can contain ESQL code and therefore support the use of ESQL variables:

- node
- node
- node
- node

Scope, lifetime, and sharing characteristics of variables

The scope, lifetime, and sharing characteristics of variables describe how widespread and for how long a particular ESQL variable is available:

Scope
Is a measure of the range over which a variable is visible. In the integration node environment, the scope of variables is typically limited to the individual node.

Lifetime
Is a measure of the time for which a variable retains its value. In the integration node environment, the lifetime of variables varies but is typically restricted to the life of a thread within a node.

Sharing characteristics
Indicate whether each thread has its own copy of a variable or whether one variable is shared between many threads. In the integration node environment, variables are typically not shared.

Types of variable

External

External variables (defined with the EXTERNAL keyword) are also known as *user-defined properties*, see "User-defined properties in ESQL" on page 1610. They exist for the entire lifetime of a message flow and are visible to all messages passing through the flow. You can define external variables only at the module and schema level. You can modify the initial values of external variables (optionally set by the DECLARE statement) at design time, by using the Message Flow editor, or at deployment time, by using the BAR editor. You can query and set the values of user-defined properties at run time by using the IBM Integration API.

Normal

Normal variables have a lifetime of just one message passing through a node. They are visible to that message only. To define normal variables, omit both the EXTERNAL and SHARED keywords.

Shared

Shared variables can be used to implement an in-memory cache in the message flow , see "Optimizing message flow response times" on page 2765. Shared variables have a long lifetime and are visible to multiple messages passing through a flow, see "Long-lived variables" on page 1611. Shared variables exist for the lifetime of the integration server process, the lifetime of the flow or node, or the lifetime of the node SQL that declares the variable (whichever is the shortest). Shared variables are initialized when the first message passes through the flow or node after each integration node startup.

See also the ATOMIC option of the BEGIN ... END statement. The BEGIN ATOMIC construct is useful when a number of changes must be made to a shared variable and it is important to prevent other instances seeing the intermediate states of the data.

User-defined properties in ESQL

Access user-defined properties (UDPs) as variables in your ESQL program by specifying the EXTERNAL keyword on a DECLARE statement. For example, the ESQL statement `DECLARE today EXTERNAL CHARACTER 'monday'` defines a user-defined property called today with an initial value monday.

Before you can use a user-defined property, you must define the property in the Message Flow editor when you construct a message flow that uses it. When you define a UDP in the Message Flow editor, you must define a value and the property type. The value can be a default value, which varies according to the type of the UDP. The value that is assigned to the UDP in the Message Flow editor takes precedence over a value that you have assigned to the UDP in your ESQL program.

You can also define a UDP for a subflow. A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.

Before you deploy the message flow that uses the UDP, you can change the value of the UDP in the BAR editor. If you try to deploy a message flow that contains a UDP that has had no value assigned to it, a deployment failure occurs. For more information, see "Configuring a message flow at deployment time with user-defined properties" on page 1749.

You can also modify the value of a UDP at run time, by using the administration REST API. For more information, see "Setting message flow user-defined properties at run time by using the administration REST API" on page 436.

You can use UDPs to set configuration data, and use them like typical properties. No external calls to user-written plug-ins or parsing of environment trees are involved, and parsing costs of reading data out of trees are removed. The value of the UDP is finalized in the variable at deployment time.

You can declare UDPs only in modules or schemas. You can query, discover, and set UDPs at run time, to dynamically change the behavior of a message flow. For more information, see "User-defined properties" on page 569.

You can access UDPs from the following built-in nodes that use ESQL:

- Compute

- Database
- DatabaseInput
- Filter

For a description of how to access a UDP from a JavaCompute node, see "Accessing message flow user-defined properties from a JavaCompute node" on page 1784.

Long-lived variables
You can use appropriate long-lived ESQL data types to cache data in memory.

Sometimes data has to be stored beyond the lifetime of a single message passing through a flow. One way to store this data is to store the data in a database. Using a database is good for long-term persistence and transactionality, but access (particularly write access) is slow.

Alternatively, you can use appropriate long-lived ESQL data types to provide an in-memory cache of the data for a certain period of time. Using long-lived ESQL data types makes access faster than from a database, although this speed is at the expense of shorter persistence and no transactionality.

You create long-lifetime variables by using the SHARED keyword on the DECLARE statement. For further information, see DECLARE statement.

Long-lived data types have an extended lifetime beyond that of a single message passing through a node. Long-lived data types are shared between threads and exist for the life of a message flow (the time between configuration changes to a message flow), as described in the following tables.

Table 66. Short lifetime variables

	Scope	**Life**	**Shared**
Schema & Module	Node	Thread within node	Not at all
Routine Local	Node	Thread within routine	Not at all
Block Local	Node	Thread within block	Not at all

Table 67. Long lifetime variables

	Scope	**Life**	**Shared**
Node Shared	Node	Life of node	All threads of flow
Flow Shared	Flow	Life of flow	All threads of flow

Features of long-lived ESQL data types include:

- The ability to handle large amounts of long-lifetime data.
- The joining of data to messages fast.
- On multiple processor machines, multiple threads can access the same data simultaneously.
- Subsequent messages can access the data left by a previous message.
- Long lifetime read-write data can be shared between threads, because there is no long-term association between threads and messages.
- In contrast to data stored in database tables in the environment, this type of data is stored privately; that is, within the integration node.
- ROW variables can be used to create a modifiable copy of the input message; see ESQL ROW data type.
- It is possible to create shared constants.

A typical use of these data types might be in a flow in which data tables are 'read-only' as far as the flow is concerned. Although the table data is not actually static, the flow does not change it, and thousands of messages pass through the flow before there is any change to the table data.

Examples include:

- A table which contains a day's credit card transactions. The table is created each day and that day's messages are run against it. Then the flow is stopped, the table updated and the next day's messages run. These flows might perform better if they cache the table data rather than read it from a database for each message.
- The accumulation and integration of data from multiple messages.

ESQL field references overview
An ESQL field reference is a sequence of period-separated values that identify a specific field (which might be a structure) within a message tree or a database table.

The path from the root of the information to the specific field is traced using the parent-child relationships.

A field reference is used in an ESQL statement to identify the field that is to be referenced, updated, or created within the message or database table. For example, you might use the following identifier as a message field reference:

```
Body.Invoice.Payment
```

You can use an ESQL variable of type REFERENCE to set up a dynamic pointer to contain a field reference. This might be useful in creating a fixed reference to a commonly-referenced point within a message; for example the start of a particular structure that contains repeating fields.

A field reference can also specify element types, XML namespace identifications, indexes and a type constraint; see ESQL field reference overview for further details.

The first name in a field reference is sometimes known as a *Correlation name*.

ESQL operators overview
An ESQL operator is a character or symbol that you can use in expressions to specify relationships between fields or values.

ESQL supports the following groups of operators:

- Comparison operators, to compare one value to another value (for example, "less than"). Refer to ESQL simple comparison operators and ESQL complex comparison operators for details of the supported operators and their use.
- Logical operators, to perform logical operations on one or two terms (for example, AND). Refer to ESQL logical operators for details of the supported operators and their use.
- Numeric operators, to indicate operations on numeric data (for example, +). Refer to ESQL numeric operators for details of the supported operators and their use.

There are some restrictions on the application of some operators to data types; not all lead to a meaningful operation. These are documented where they apply to each operator.

Operators that return a Boolean value (TRUE or FALSE), for example the "greater than" operator, are also known as *predicates*.

ESQL statements overview
An ESQL statement is an instruction that represents a step in a sequence of actions or a set of declarations.

ESQL provides a large number of different statements that perform different types of operation. All ESQL statements start with a keyword that identifies the type of statement and end with a semicolon. An ESQL program consists of a number of statements that are processed in the order they are written.

As an example, consider the following ESQL program:

```
DECLARE x INTEGER;
SET x = 42;
```

This program consists of two statements. The first starts with the keyword DECLARE and ends at the first semicolon. The second statement starts with the keyword SET and ends at the second semicolon. These

two statements are written on separate lines and it is conventional (but not required) that they be so. You will notice that the language keywords are written in capital letters. This is also the convention but is not required; mixed case and lowercase are acceptable.

The first statement declares a variable called x of type INTEGER, that is, it reserves a space in the computer's memory large enough to hold an integer value and allows this space to be subsequently referred to in the program by the name x. The second statement sets the value of the variable x to 42. A number appearing in an ESQL program without decimal point and not within quotation marks is known as an integer literal.

ESQL has a number of data types and each has its own way of writing literal values. These are described in “ESQL data types overview” on page 1608.

For a full description of all the ESQL statements, see ESQL statements.

ESQL nested statements
An ESQL nested statement is a statement that is contained within another statement.

Consider the following ESQL program fragment:

```
IF Size> 100.00 THEN
  SET X = 0;
  SET Y = 0;
  SET REVERSE = FALSE;
ELSE
  SET X = 639;
  SET Y = 479;
  SET REVERSE = TRUE;
END IF;
```

In this example, you can see a single IF statement containing the optional ELSE clause. Both the IF and ELSE portions contain three nested statements. Those within the IF clause are processed if the operator > (greater than) returns the value TRUE (that is, if Size has a value greater than 100.00); otherwise, those within the ELSE clause are processed.

Many statements can have expressions nested within them, but only a few can have statements nested within them. The key difference between an expression and a statement is that an expression calculates a value to be used, whereas a statement performs an action (usually changing the state of the program) but does not produce a value.

ESQL functions overview
A function is an ESQL construct that calculates a value from a number of given input values.

A function usually has input parameters and can, but does not usually have, output parameters. It returns a value calculated by the algorithm described by its statement. This statement is usually a compound statement, such as BEGIN... END, because this allows an unlimited number of nested statements to be used to implement the algorithm.

ESQL provides a number of predefined, or "built-in", functions which you can use freely within expressions. You can also use the CREATE FUNCTION statement to define your own functions.

When you define a function, you must give it a unique name. The name is handled in a non-case sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). This is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

Consider the following ESQL program fragment:

```
SET Diameter = SQRT(Area / 3.142) * 2;
```

In this example, the function SQRT (square root) is given the value inside the brackets (itself the result of an expression, a divide operation) and its result is used in a further expression, a multiply operation. Its return value is assigned to the variable Diameter. See Calling ESQL functions for information about all the built-in ESQL functions.

In addition, an ESQL expression can refer to a function in another broker schema (that is, a function defined by a CREATE FUNCTION statement in an ESQL file in the same or in a different dependent project). To resolve the name of the called function, you must take one of the following steps:

- Specify the fully-qualified name (<SchemaName>.<FunctionName>) of the called function.
- Include a PATH statement to make all functions from the named schema visible. Note that this technique only works if the schemas do not contain identically-named functions. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Note that you cannot define a function within an EVAL statement or an EVAL function.

ESQL procedures overview
An ESQL procedure is a subroutine that has no return value. It can accept input parameters from, and return output parameters to, the caller.

Procedures are very similar to functions. The main difference between them is that, unlike functions, procedures have no return value. Thus they cannot form part of an expression and are invoked by using the CALL statement. Procedures commonly have output parameters

You can implement a procedure in ESQL (an internal procedure) or as a database stored procedure (an external procedure). The ESQL procedure must be a single ESQL statement, although that statement can be a compound statement such as BEGIN END. You cannot define a procedure within an EVAL statement or an EVAL function.

When you define a procedure, give it a name. The name is handled in a non-case sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). That is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case sensitive way, and which you must specify exactly as you declared them.

An ESQL expression can include a reference to a procedure in another broker schema (defined in an ESQL file in the same or a different dependent project). If you want to use this technique, either fully qualify the procedure, or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within a MODULE.

An external database procedure is indicated by the keyword EXTERNAL and the external procedure name. This procedure must be defined in the database and in the integration node, and the name specified with the EXTERNAL keyword and the name of the stored database procedure must be the same, although parameter names do not have to match. The ESQL procedure name can be different from the external name it defines.

Overloaded procedures are not supported to any database. (An overloaded procedure is one that has the same name as another procedure in the same database schema which has a different number of parameters, or parameters with different types.) If the integration node detects that a procedure has been overloaded, it raises an exception.

Dynamic schema name resolution for stored procedures is supported; when you define the procedure you must specify a wildcard for the schema that is resolved before invocation of the procedure by ESQL. This is explained further in "Invoking stored procedures" on page 1674.

ESQL modules overview
A module is a sequence of declarations that define variables and their initialization, and a sequence of subroutine (function and procedure) declarations that define a specific behavior for a message flow node.

A module must begin with the `CREATE node_type MODULE` statement and end with an `END MODULE` statement. The *node_type* must be one of COMPUTE, DATABASE, DATABASEEVENT, or FILTER. For DatabaseInput nodes, the required *node_type* is DATABASEEVENT. For Compute, Database, and Filter nodes the required *node_type* is the same as the node. For example, you must use the COMPUTE *node_type* in ESQL modules for Compute nodes. For Compute, Database and Filter nodes, the entry point of the ESQL code is the function named MAIN, which has MODULE scope. ESQL modules for DatabaseInput nodes do not contain the MAIN function, but contain three procedures named READEVENTS, BUILDMESSAGE, and ENDEVENT. For more information about these three procedures, see "Configuring a DatabaseInput node" on page 1140.

Each module is identified by a name which follows CREATE *node_type* MODULE. The name might be created for you with a default value, which you can modify, or you can create it yourself. The name is handled in a non-case-sensitive way (that is, use of the name with any combination of uppercase and lowercase letters matches the declaration). This is in contrast to the names that you declare for schemas, constants, variables, and labels, which are handled in a case-sensitive way, and which you must specify exactly as you declared them.

You must create the code for a module in an ESQL file which has a suffix of `.esql`. You must create this file in the same broker schema as the node that references it. There must be one module of the correct type for each corresponding node, and it is specific to that node and cannot be used by any other node.

When you create an ESQL file (or complete a task that creates one), you indicate the integration project and broker schema with which the file is associated and specify a name for the file.

Within the ESQL file, the name of each module is determined by the value of the corresponding property of the message flow node. For example, the property *ESQL Module* for the Compute node specifies the name of the module in the ESQL file for that node. The default value for this property is the name of the node. You can specify a different name, but you must ensure that the value of the property and the name of the module that provides the required function are the same.

For Compute, Database, and Filter nodes, the module must contain the function MAIN, which is the entry point for the module. This function is included automatically if the module is created for you. Within MAIN, you can code ESQL to configure the behavior of the node. If you include ESQL within the module that declares variables, constants, functions, and procedures, these are of local scope only and can be used within this single module.

For DatabaseInput nodes, the module must contain the three procedures READEVENTS, BUILDMESSAGE, and ENDEVENT. These procedures are included automatically, together with comments to describe the procedure, if the module is created for you. Within the three procedures, you can code ESQL to configure the behavior of the node, see "Configuring a DatabaseInput node" on page 1140. If you include ESQL within the module that declares variables, constants, functions, and procedures, these are of local scope only and can be used within this single module.

If you want to reuse ESQL constants, functions, or procedures, you must declare them at broker schema level. You can then refer to these constants, functions or procedures from any resource within that broker schema, in the same or another project. If you want to use this technique, either fully qualify the procedure, or include a PATH statement that sets the qualifier. The PATH statement must be coded in the same ESQL file, but not within any MODULE.

Managing ESQL files

In an application, library, or integration project, manage ESQL files that contain the ESQL code that you provide to modify or customize the behavior of Compute, Database, DatabaseInput, and Filter nodes.

About this task

The ESQL code for a message flow node is contained in a module that is associated with the node. Each module must be created in an ESQL file (file extension `.esql`) . You can change the name of the module in the ESQL file from its default value of *MessageFlow_NodeName*, where *MessageFlow* is the name of the message flow and *NodeName* is the name of the message flow node. The name of the module in the ESQL file must match the name specified for the module in the ESQL `Module` property of the corresponding message flow node.

By default, ESQL files are deployed to the integration server as individual resources, and can be edited and redeployed without redeploying the message flows that reference them. Alternatively, you can include the code from an ESQL file directly in the compiled message flow (`.cmf` file) that references it. Compiled message flows that include ESQL code in this manner must be redeployed each time you change the ESQL code. For more information, see "Adding resources to a BAR file" on page 2470.

The following topics describe how to manage ESQL files.

- "Creating an ESQL file" on page 1616
- "Opening an existing ESQL file" on page 1617

- “Creating ESQL for a node” on page 1618
- “Deploying an ESQL file” on page 1623
- “Modifying ESQL for a node” on page 1620
- “Saving an ESQL file” on page 1621
- “Copying an ESQL file” on page 1623
- “Renaming an ESQL file” on page 1624
- “Moving an ESQL file” on page 1624
- “Changing ESQL preferences” on page 1625
- “Deleting ESQL for a node” on page 1627
- “Deleting an ESQL file” on page 1627

Creating an ESQL file
When you include a node in your message flow that requires ESQL code to customize its function, you must code the ESQL statements that provide the customization in an ESQL module in an ESQL file. The Compute, Database, DatabaseInput, and Filter nodes use ESQL code. You can use the same ESQL file for more than one module.

Before you begin

For background information, see “ESQL overview” on page 1608.

Optional: You can create an application, library, or integration project before you create an ESQL file. Alternatively, you can create the containing project when you create the ESQL file.

About this task

ESQL files are stored in a file system or in a shared repository. If you are using a file system, you can use the local file system or a shared drive. If you store files in a repository, you can use any of the available repositories that are supported by Eclipse, for example CVS.

To create an ESQL file, complete the following steps.

Procedure

1. Click **File** > **New** > **Message Flow ESQL File**.

   You can also click **New** in the Application Development view, then click **Message Flow ESQL File**.

   The **New Message Flow ESQL File** wizard opens.
2. Either select an existing application, library, or integration project in which to create the ESQL file, or click **New** to create a new container.

   When you store ESQL files in a shared library, you must place the ESQL files inside a schema that is not the default empty schema. For more information, see “Shared libraries” on page 1948.
3. Enter a name for the new ESQL file.

   If you enter a name that is already in use for an ESQL file in this project, the error message The `resource <name>.esql already exists` is shown and you must specify a different name.

   When creating ESQL files, the overall file path length must not exceed 256 characters, due to a Windows file system limitation. If you try to add a message flow to a BAR file with ESQL or mapping files with a path length that exceeds 256 characters, the compiled message flow is not generated and cannot be deployed. Therefore, make sure that the names of your ESQL files, mapping files, projects, and broker schema are as short as possible.
4. To define the ESQL file in a specific broker schema, clear **Use default broker schema** and either select a broker schema from the drop-down list, or enter the name of the broker schema.
5. Click **Finish**.

Results

The ESQL file opens in the editor, where you can edit the file, then save it. The ESQL file is shown in the Application Development view, beneath the ESQLs folder of the containing application, library, or integration project.

An ESQL file can also be created automatically for you. You can right-click a Compute, Database, DatabaseInput, or Filter node, then click **Open ESQL**. If the module identified by the appropriate property does not already exist in the broker schema, a module is created automatically. This module is created in the file `<message_flow_name>.esql` in the same broker schema in the same project as the `<message_flow_name>.msgflow` file. If that ESQL file does not already exist, that is also created for you.

The contents of a single ESQL file do not have any specific relationship with message flows and nodes. It is your decision which modules are created in which files (unless the specified module, identified by the appropriate property, is created by default in the file `<message_flow_name>.esql` as described above). Monitor the size and complexity of the ESQL in each file, and split the file if it becomes difficult to view or manage.

If you create reusable subroutines (at broker schema level) in an ESQL file, you might want to refer to these routines from ESQL modules in another project. To refer to the routines, specify that the project that runs the subroutines depends on the project in which the ESQL file containing them is defined. You can specify this behavior when you create the second project, or you can create a project reference; see "Referencing resources in other libraries" on page 1976.

Opening an existing ESQL file
You can add to and modify ESQL code that you have created in an ESQL file.

Before you begin

To complete this task, you must have created an ESQL file, as described in "Creating an ESQL file" on page 1616.

About this task

To open an existing ESQL file, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application, library, or integration project, expand the ESQLs folder, then double-click the ESQL file that you want to open.

   The ESQL file opens in the editor.
2. Edit the contents of file to make your changes.

   The file can contain modules that relate to specific nodes in a message flow, PATH statements, and declarations at broker schema level, such as reusable constants and procedures. You can select the content that you want to work with by selecting its name in the Outline view. The code for the selected resource is highlighted.
3. Save and close the ESQL file.

Results

You can also open an ESQL file when you have a message flow open in the editor view by right-clicking an appropriate node (a Compute, Database, DatabaseInput, or Filter node), then clicking **Open ESQL**. In this case, the ESQL file that contains this module is opened, and the module for the selected node is highlighted in the editor view.

Creating ESQL for a node
Create ESQL code to customize the behavior of a Compute, Database, DatabaseInput, or Filter node in an ESQL file.

Before you begin

This topic assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 1616.

About this task

In the ESQL file, create a module that is associated with a node in your message flow. A module can be associated with only one node of a particular type (Compute, Database, DatabaseInput, or Filter). Within the module, you can create and use functions and procedures in addition to the supplied statements and functions. You can also create local constants and variables.

If you have created constants, functions, or procedures at the broker schema level, you can also refer to these in the module. You can define routines at a level at which many different modules can use them, which can save you development time and maintenance effort.

To create ESQL for a node, complete the following steps.

Procedure

1. Open the message flow that includes the node for which you want to create ESQL.

   In the Application Development view, expand the appropriate application, library, or integration project, expand the Flows folder, then double-click the message flow.

   The message flow opens in the Message Flow editor.
2. Right-click a Compute, Database, DatabaseInput, or Filter node, then click **Open ESQL**.

   The default ESQL file for this message flow, *message_flow_name*.esql, opens in the editor view. If the file does not already exist, it is created, containing a skeleton module for this node at the end. The exact content depends on the type of node.

   If you have already created the file, it is opened in the editor view and a new module is created and highlighted.

   The following module is created for a Compute node:

```
CREATE COMPUTE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
              -- CALL CopyMessageHeaders();
              -- CALL CopyEntireMessage();
              RETURN TRUE;
       END;

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

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

   The module name is determined by the value that you have set for the corresponding node property. The default name is *message_flow_name_node_type*. The main function contains calls to two procedures (described in the following list) that are declared in the Compute node module following

the function Main. These calls are commented out. To include the function that they provide, uncomment the lines and place them at the appropriate point in the ESQL that you create for Main.

CopyMessageHeaders
This procedure loops through the headers contained in the input message and copies each one to the output message.

CopyEntireMessage
This procedure copies the entire contents of the input message, including the headers, to the output message.

If you create an ESQL module for a Database node, the following module is created:

```
CREATE DATABASE MODULE module_name
       CREATE FUNCTION Main() RETURNS BOOLEAN
       BEGIN
                RETURN TRUE;
       END;
END MODULE;
```

For a DatabaseInput node, the module that is created contains three procedures, ReadEvents, BuildMessage, and EndEvent. Each of these procedures contains boilerplate text, which describes how the procedure works. For more information about configuring DatabaseInput nodes, see “Configuring a DatabaseInput node” on page 1140. For a DatabaseInput node, the following first line of the module is created:

```
CREATE DATABASEEVENT MODULE module_name
```

For a Filter node, the module is identical to the module created for the Database node, except for the first line, which reads:

```
CREATE FILTER MODULE module_name
```

3. Add ESQL to this file to customize the behavior of the node.

   For Compute, Database, or Filter nodes, start by adding ESQL statements in the Main function, (after the BEGIN statement, and before RETURN TRUE). For DatabaseInput nodes, add ESQL statements in the ReadEvents, BuildMessage, and EndEvent procedures. You can add DECLARE statements in the module that are not within the Main function. To add a new line into the file, press Enter.

   To help you to code valid ESQL, the editor shows a list of valid statements and functions at the point of the cursor. To start this assistance, click **Edit** > **Content Assist**. On some systems, you can use the key combination Ctrl+Space. Scroll through the list to find and highlight the statement or function that you want, and press Enter. The appropriate code is inserted into your module.

   Content assistance is provided in the following areas:

   - Applicable keywords, based on language syntax.
   - Blocks of code that go together, such as `BEGIN END;`.
   - Constants that you have defined, identifiers, labels, functions, and procedures that can be used, where the routines can be in any projects, even if the current project does not reference them.
   - Database schema and table names after the database correlation name, table column names in INSERT, UPDATE, DELETE, and SELECT statements, and, in most cases, the WHERE clauses of those statements.
   - Elements of message field reference: runtime domain (parser) names, format of type expression, namespace identifiers, namespace-qualified element and attribute names, and format of index expression.
   - Content in the Properties folder under the output message.
   - For the DECLARE NAMESPACE statement, target namespaces of message sets and schema names.

   Content assistance works only if the ESQL can be parsed correctly. Errors such as END missing after BEGIN, and other unterminated block statements, cause parser failures and no content assistance

is provided. Try content assistance in other areas around the statement where it does not work to narrow down the point of error. Alternatively, save the ESQL file; saving the file causes validation, and all syntax errors are written to the Problems view. Refer to the errors reported to understand and correct the ESQL syntax. If you use content assistance to generate most statements (such as block statements), these statements are correctly entered and there is less opportunity for error.

4. When you have finished working with this module, save and close the ESQL file.

Results

You can also open the ESQL file directly and create the module in that file by using the editor:

1. Open the ESQL file in which you want to create the module.
2. In the editor view, position your cursor on a new line and use content assistance to select the appropriate module skeleton for this type of node, for example `CREATE COMPUTE MODULE END MODULE;`. You can also type in this text, but you must ensure that what you type is consistent with the required skeleton, shown earlier. Use content assistance to give you additional help by inserting only valid ESQL, and by inserting matching end statements (for example, `END MODULE;`) where they are required.
3. Complete the coding of the module as appropriate.

Whichever method you use to open the ESQL file, be aware that the editor provides functions to help you to code ESQL. This section refers to content assistance; other functions are available. For information about these functions, see ESQL editor.

Modifying ESQL for a node
To change the customization of a node that requires ESQL (Compute, Database, DatabaseInput, or Filter node), modify the ESQL statements in the module that you created for that node.

Before you begin

This topic assumes that you have created ESQL code for a file. For more information, see “Creating ESQL for a node” on page 1618.

About this task

To modify ESQL code, complete the following steps.

Procedure

1. Open the message flow with which you want to work.

   In the Application Development view, expand the appropriate application, library, or integration project, expand the Flows folder, then double-click the message flow.)

   The message flow is opened in the Message Flow editor.
2. Right-click the node that corresponds to the ESQL module that you want to modify and click **Open ESQL**.

   The ESQL file is opened in the editor view. The module for this node is highlighted.
3. Make the required changes in the module by entering new statements, changing existing statements by over-typing, or deleting statements by using the Delete or backspace keys. You can use Content Assist, available from the **Edit** menu or, on some systems, by pressing Ctrl+Space). To get Content Assist to work with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see “Referencing resources in other libraries” on page 1976.
4. You can change the name of the module with which you are working by over-typing the current name with the new one. In this case, you must also change the node property ESQL `Module` to reflect the new name, to ensure that the correct ESQL code is deployed with the node.
5. When you have finished working with this module, save and close the ESQL file.

Results

You can also open the ESQL file directly by double-clicking it in the Application Development view. You can select the module with which you want to work from the Outline view.

The editor provides functions that you can use to help you modify your ESQL code. These functions are described in ESQL editor.

You can also modify the ESQL source by clicking **Source** > **Format**. This option formats all selected lines of code (unless only partially selected, when they are ignored), or, if no lines are selected, formats the entire file (correcting alignments and indentation).

Adding comments to ESQL

About this task

You can add and remove comments in your ESQL code.

Procedure

- To change an existing line of code into a comment line, click **Source** > **Comment**.
- To change a comment line to a code line, click **Source** > **Uncomment**.
- To create a new comment line, press Enter to create a new line, then either type the comment identifier `--` or click **Source** > **Comment**. You can enter any text after the identifier; everything that you type is ignored by the ESQL editor.

Saving an ESQL file
When you edit an ESQL file, you can save it to preserve the additions and modifications that you have made, and to force the editor to validate the content of the file.

Before you begin

This task assumes that you have created or opened an ESQL file. For more information, see “Creating an ESQL file” on page 1616 or “Opening an existing ESQL file” on page 1617.

About this task

To save an ESQL file, complete the following steps.

Procedure

1. Change the contents of the ESQL file.

2. Save the file by clicking **File** > **Save** or **File** > **Save All**.

   When you save the file, the validator is called by the editor to check that the ESQL obeys all grammar and syntax rules (specified by the syntax diagrams and explanations in ESQL reference).

   You can request additional validation when you set ESQL preferences:

   a) Click **Window** > **Preferences** > **Integration Development** > **ESQL** > **Validation**.
   The Preferences dialog box is displayed.

   b) Select the level of validation that you require for each category of error:

      i) Unresolved identifiers

      ii) Message references do not match message definitions

      iii) Database references do not match database schema

      iv) Use of deprecated keywords

      The default level is *warning*; you can change this value to *error* or *ignore*.

      Validating message definitions can affect response times in the editor, particularly if you have complicated ESQL that makes many references to a complex message definition. You might choose to delay this validation. Call validation when you have finished developing the message flow and are about to deploy it, to avoid runtime errors.

      For each error found, the editor writes the code line number and the reason for the error; errors are created as entries in the Problems view.

3. If you double-click the error, the editor positions your cursor on the line in which it found that error.

   The line is also highlighted by the error icon  .

   The editor might also find potential error situations, which it highlights as warnings (with the warning icon ); the editor also writes these warnings to the Problems view. For example, you might have included a BROKER SCHEMA statement that references an invalid schema (namespace).

   Check your code, and make the corrections required by that statement or function.

Save As

About this task

You can save a copy of this ESQL file by using **File** > **Save As**.

Procedure

1. Click **File** > **Save As**.
2. Specify the integration project in which you want to save a copy of the ESQL file.

   The project name defaults to the current project. You can accept this name, or choose another name from the valid options that are displayed in the File Save dialog box.
3. Specify the name for the new copy of the ESQL file.

   To save this ESQL file in the same project, either rename it, or confirm that you want to overwrite the current copy (that is, copy the file to itself).

   To save this ESQL file in another project, the project must exist. You can save the file with the same or another name in another project.
4. Click **OK**.

   The message flow is saved and the message flow editor validates its contents. The editor provides a report of all errors that it finds in the Problems view.

Copying an ESQL file
You can copy an ESQL file as a starting point for a new ESQL file that has similar function.

Before you begin

This topic assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 1616.

About this task

To copy an ESQL file, complete the following steps.

Procedure

1. In the Application Development view, right-click the ESQL file that you want to copy, and click **Copy**.

   Alternatively, you can select the ESQL file and click **Edit** > **Copy**.
2. Right-click the integration project into which you want to copy the ESQL file and click **Paste**.

   Alternatively, you can select the integration project and click **Edit** > **Paste**.

   You can copy the ESQL file to the same broker schema within the same integration project, or to a different broker schema within the same integration project, or to a broker schema in a different integration project.

   When you copy an ESQL file, the associated files (message flow, and mapping if present) are not automatically copied to the same target integration project. If you want these files copied as well, you must do so manually.

   To use this ESQL file with another message flow, ensure that the modules in the ESQL file match the nodes that you have in the message flow, and that the node properties are set correctly.

Results

You can also copy an ESQL file by using **File** > **Save As**; for more information, see "Saving an ESQL file" on page 1621.

Deploying an ESQL file
By default, ESQL files are deployed in BAR files as individual resources, and can be edited and redeployed without redeploying the message flows that reference them.

Before you begin

This topic assumes that you have created an ESQL file. For more information, see "Creating ESQL for a node" on page 1618.

About this task

When you create a BAR file and add your message flow, the IBM App Connect Enterprise Toolkit packages any ESQL files that are referenced by the message flow into the BAR file. If the ESQL code references additional ESQL files that are not directly referenced by the message flow, the additional ESQL files must be added to the BAR file manually before it is deployed.

Optionally, you can include the code from an ESQL file directly in the compiled message flow (`.cmf` file) that references it. Compiled message flows that include ESQL code in this manner must be redeployed each time you change the ESQL code. For more information, see "Adding resources to a BAR file" on page 2470.

Deployed ESQL files are displayed in the Integration Explorer view. If your messaging solution uses applications or libraries, deployed ESQL files are displayed under the application or library to which they

were deployed. If your messaging solution does not use applications or libraries, deployed ESQL files are displayed under the integration server to which they were deployed.

For more information, see:

- "Creating a BAR file" on page 2469
- "Adding resources to a BAR file" on page 2470
- "Deploying integration solutions to a production environment" on page 2480

Renaming an ESQL file
You might want to rename an ESQL file if you have renamed the message flow with which it is associated.

Before you begin

This task assumes that you have created an ESQL file, as described in "Creating an ESQL file" on page 1616.

About this task

To rename an ESQL file, complete the following steps.

Procedure

1. Locate the ESQL file that you want to rename by expanding the appropriate application, library, or integration project, then expanding the ESQLs folder.
2. Right-click the ESQL file and click **Rename**.

   Alternatively, you can select the ESQL file, then click **File** > **Rename**.

   The Rename Resource dialog box opens.
3. Enter the new name for the ESQL file, then click **OK**.

   The ESQL file is renamed.

Moving an ESQL file
If you move a message flow from one broker schema to another, or from one project to another, you might want to move all ESQL files that are associated with that message flow.

Before you begin

To complete this task, you must have completed the following task:

- "Creating an ESQL file" on page 1616

About this task

To move an ESQL file:

Procedure

1. Move the ESQL file in one of the following ways:
   - Drag the ESQL file that you want to move from its current location to a broker schema within the same or another integration project.

     If the target location that you have chosen is not valid (for example, if an ESQL file of this name exists in the broker schema), the invalid icon is displayed and the move is not completed. If you have created an empty broker schema for this purpose, it might not be visible in the Application

Development view if category mode is selected. To see an empty schema in the Application Development view, click Hide Categories  .

- Right-click the ESQL file and click **Move**, or click **File** > **Move**. The Move dialog is displayed.

  Select the project and the broker schema from the list of valid targets that is shown in the dialog.

  Click **OK** to complete the move, or **Cancel** to cancel the request.

  If you click **OK**, the ESQL file is moved to its new location.

2. Check the Problems view for errors (indicated by the error icon ) or warnings (indicated by the warning icon ) generated by the move.

   The errors in the Problems view include those errors caused by broken references. When the move is completed, all references to this ESQL file are checked. If you have moved the file within the same named broker schema within the same integration project, all references are still valid. If you have moved the file to another broker schema in the same or another integration project, the references are broken. If you have moved the file to the same named broker schema in another integration project, the references might be broken if the project references are not set correctly to recognize external references in this file. These errors occur because resources are linked by a fully qualified name.

3. Double-click each error or warning to open the message flow that has the error in the editor view, and highlight the node in error.

   You can now correct the error.

Results

When you move an ESQL file, its associated files (for example, the message flow file) are not automatically moved to the same target broker schema. You must move these files yourself.

Changing ESQL preferences
You can modify the way in which ESQL is displayed in the editor and validated by the editor:

Procedure

- “Changing ESQL editor settings” on page 1626
- “Changing ESQL validation settings” on page 1626

Changing ESQL editor settings
When you open an ESQL file in the editor view, you can tailor the editor appearance by changing editor settings.

Procedure

1. **To change ESQL editor settings:**
   a) Click **Window** > **Preferences**.
   The Preferences dialog box opens.
   b) Expand the item for ESQL on the left and click **ESQL Editor**.
   c) Update the settings available for tab width and colors:
   - Click the **General** tab to change the displayed tab width within the ESQL editor.
   - Click the **Colors** tab to change the color of the editor view background, and of the entities displayed in the editor view. These include comments and keywords in your ESQL code.

   d) When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open), or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
   e) To return the ESQL editor settings to the initial values, click **Restore Defaults**.
   All values are reset to the original settings.

   If you change the editor settings when you have an editor session active, the changes are implemented immediately. If you do not have an editor session open, you see the changes when you next edit an ESQL file.
2. **To change font settings for the ESQL editor:**
   a) Click **Window** > **Preferences**. The Preferences dialog box opens.
   b) Expand the item for Workbench on the left of the Preferences dialog box, and click **Colors and Fonts**.
   c) On the **Colors and Fonts** tab, expand Basic.
   d) Select a font or text color option and click **Change**. The Font dialog box opens.
   e) When you have completed your changes, click either **Apply** (to apply your changes and leave the Preferences dialog box open) or **OK** (to apply your changes and close the dialog box). Alternatively, to close the dialog box and discard your changes, click **Cancel**.
   f) To return the ESQL editor settings to the initial values, click **Restore Defaults**.

Changing ESQL validation settings
You can specify the level of validation that the ESQL editor performs when you save a `.esql` file. If the validation you have requested results in warnings, you can deploy a BAR file containing this message flow. However, if errors are reported, you cannot deploy the BAR file.

About this task

To change ESQL validation settings:

Procedure

1. Switch to the Integration Development perspective.
2. Click **Window** > **Preferences**.
   The Preferences dialog is displayed.
3. Expand the item for ESQL on the left and click Validation.
4. Update the settings for what is validated, and for what warnings or errors are reported.
   See ESQL editor for details of the settings and their values.

5. When you have completed your changes, click **Apply** to close the Preferences dialog, apply your changes and leave the Preferences dialog open. Click **OK** to apply your changes and close the dialog. Click **Cancel** to close the dialog and discard your changes.
6. If you want to return your ESQL editor preferences to the initial values, click **Restore Defaults**.

   All values are reset to the original settings.

Results

If you make changes to the validation settings, the changes are implemented immediately for currently open edit sessions and for subsequent edit sessions.

Deleting ESQL for a node
If you delete a node from a message flow, you can delete the ESQL module that you created to customize its function.

Before you begin

This topic assumes that you have created ESQL code for a node. For more information, see "Creating ESQL for a node" on page 1618.

About this task

To delete ESQL code, complete the following steps.

Procedure

1. Open the message flow with which you want to work.

   Expand the appropriate application, library, or integration project, expand the Flows folder, then double-click the message flow.

   The message flow opens in the Message Flow editor.
2. Right-click the node for which you want to delete the ESQL module, then click **Open ESQL**.

   The ESQL file is opened in the editor view, with the module for this node highlighted.
3. Press the Delete or backspace key to delete the whole module.
4. When you have finished working with this module, save and close the ESQL file.

   Saving the file also validates the ESQL; for more details, see "Saving an ESQL file" on page 1621.

Deleting an ESQL file
If you delete a message flow, or if you have deleted all the ESQL code in an ESQL file, you can delete the ESQL file.

Before you begin

This task assumes that you have created an ESQL file. For more information, see "Creating an ESQL file" on page 1616.

About this task

To delete an ESQL file, complete the following steps.

Procedure

1. In the Application Development view, right-click the ESQL file that you want to delete, and click **Delete**.

   You can also select the file in the Application Development view, and click **Edit** > **Delete**.

   A dialog box is displayed that asks you to confirm the deletion.

2. Click **Yes** to delete the file, or **No** to cancel the delete request.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the file if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the file is retained. Be careful to select the correct file when you complete this task.

Exporting mapping information from Compute nodes
Export information for use by products, such as IBM InfoSphere® Metadata Workbench, that can perform impact analysis and data lineage operations.

The **Data Lineage Documents** wizard can generate the following documents:

Extension mapping documents
Extension mapping documents are comma-separated files that contain information about mapping transformations from Compute nodes in an IBM App Connect Enterprise project. You can generate mapping documents for all message flows, or for a single flow.

Extended data source documents
Extended data source documents describe the data structure of messages from each message set referenced by an IBM App Connect Enterprise project.

Procedure

1. Use one of the following methods to start the **Data Lineage Documents** wizard:
   - Right-click the application, library, or project, and then select **New** > **Data Lineage Documents**.
   - Click **File** > **New** > **Data Lineage Documents**.
2. Complete the wizard as appropriate.

Data lineage CSV files
Comma-separated value (CSV) files for use in impact analysis and data lineage.

You can export all mapping transformations from all Compute nodes in a specified application, library, or IBM App Connect Enterprise project. You can also create extended data source documents for schemas in a message set project.

The resulting CSV files can be imported into products, such as IBM InfoSphere Metadata Workbench, that can perform impact analysis and data lineage operations.

Data lineage
See where the information comes from, and what happens to it as it moves across data integration processes.

Impact analysis
Determine the effects of changing a field in the information flow.

The **Data Lineage Documents** wizard creates the following artifact as CSV files:

Extension Mapping Documents
This artifact contains rows of mappings from IBM App Connect Enterprise transformations between message models in a message flow. One mapping document is created for each flow, containing all mappings done by different Compute nodes in the flow. Each transformation in a node is recorded on its own row:

```
Name,Source Columns,Target Columns,Rule,Function,Specification Description
"Compute","WMBMessageModels.ShipOrderLibLibrary.shipPartRequest.shipPartRequest/
partNo","WMBMessageModels.ShipOrderLibLibrary.shipPartResponse.shipPartResponse/
partNo",,,"{}ProcessShipping_Compute.Main"
"Compute","WMBMessageModels.ShipOrderLibLibrary.shipPartRequest.shipPartRequest/
partQuantity","WMBMessageModels.ShipOrderLibLibrary.shipPartResponse.shipPartResponse/
partQuantity",,,"{}ProcessShipping_Compute.Main"
```

The fields are as follows:

Name
The name of the node in the message flow.

Source Columns
The source of the mappings.

Target Columns
The target of the mappings.

Rule
Not currently used.

Function
A list of user-defined or built-in functions that are being called in the mapping. If the list exceeds 255 characters, it is truncated.

Specification Description
The Compute node module or ESQL procedure or function where the mapping takes place, that is where the target is being mapped. If it is in a function or procedure inside a compute module, it has this form:

```
{broker.schema1}MyMessageFlow2_Compute.Main, {}Compute2.CopyEntireMessage.
```

If it is a procedure or function outside the module scope, the procedure or function name is shown, for example:

```
{broker.schema1}ExternalProcedure1
```

Not all ESQL transformations are supported; see Data Lineage restrictions.

Writing ESQL

How you can use ESQL to customize nodes.

About this task

When you create a message flow, you include input nodes that receive the messages and, optionally, output nodes that send out new or updated messages. If required by the processing that must be performed on the message, you can include other nodes after the input node that complete the actions that your applications need.

Some of the built-in nodes enable you to customize the processing that they provide. The Compute, Database, DatabaseInput, and Filter nodes require you to provide a minimum level of ESQL, and you can provide much more than the minimum to control precisely the behavior of each node. This set of topics discusses ESQL and the ways in which you can use it to customize these nodes.

You can use the Mapping node to customize the processing visually. You can also call ESQL from a map to process or reuse existing or common ESQL routines. For more information, see Using message maps

ESQL provides a rich and flexible syntax for statements and functions that enable you to check and manipulate message and database content. You can:

- Read the contents of the input message
- Modify message content with data from databases
- Modify database content with data from messages
- Construct new output messages created from all, part, or none of the input message (in the Compute node only)

The following topics provide more information about these and other tasks that you can perform with ESQL. Unless otherwise stated, these guidelines apply to messages in all message domains except the BLOB domain, for which you can implement a limited set of actions.

- “Tailoring ESQL code for different node types” on page 1631
- “Manipulating message body content” on page 1632

- “Manipulating other parts of the message tree” on page 1649
- “Transforming from one data type to another” on page 1659
- “Interaction with databases using ESQL” on page 1666
- “Coding ESQL to handle errors” on page 1676
- “Configuring a message flow at deployment time with user-defined properties” on page 1749
- “Accessing a user-defined policy by using ESQL” on page 1750

The following topics provide additional information specific to the parser that you have specified for the input message:

- “Manipulating messages in the DFDL domain” on page 1693
- “Manipulating messages in the XML domain” on page 1727
- “Manipulating messages in the MRM domain” on page 1727
- “Manipulating messages in the JMS domains” on page 1741
- “Manipulating messages in the XMLNS domain” on page 1719
- “Manipulating messages in the XMLNSC domain” on page 1706
- “Manipulating messages in the IDOC domain” on page 1741
- “Manipulating messages in the MIME domain” on page 1742
- “Manipulating messages in the BLOB domain” on page 1744
- “Manipulating messages in the JSON domain” on page 1744

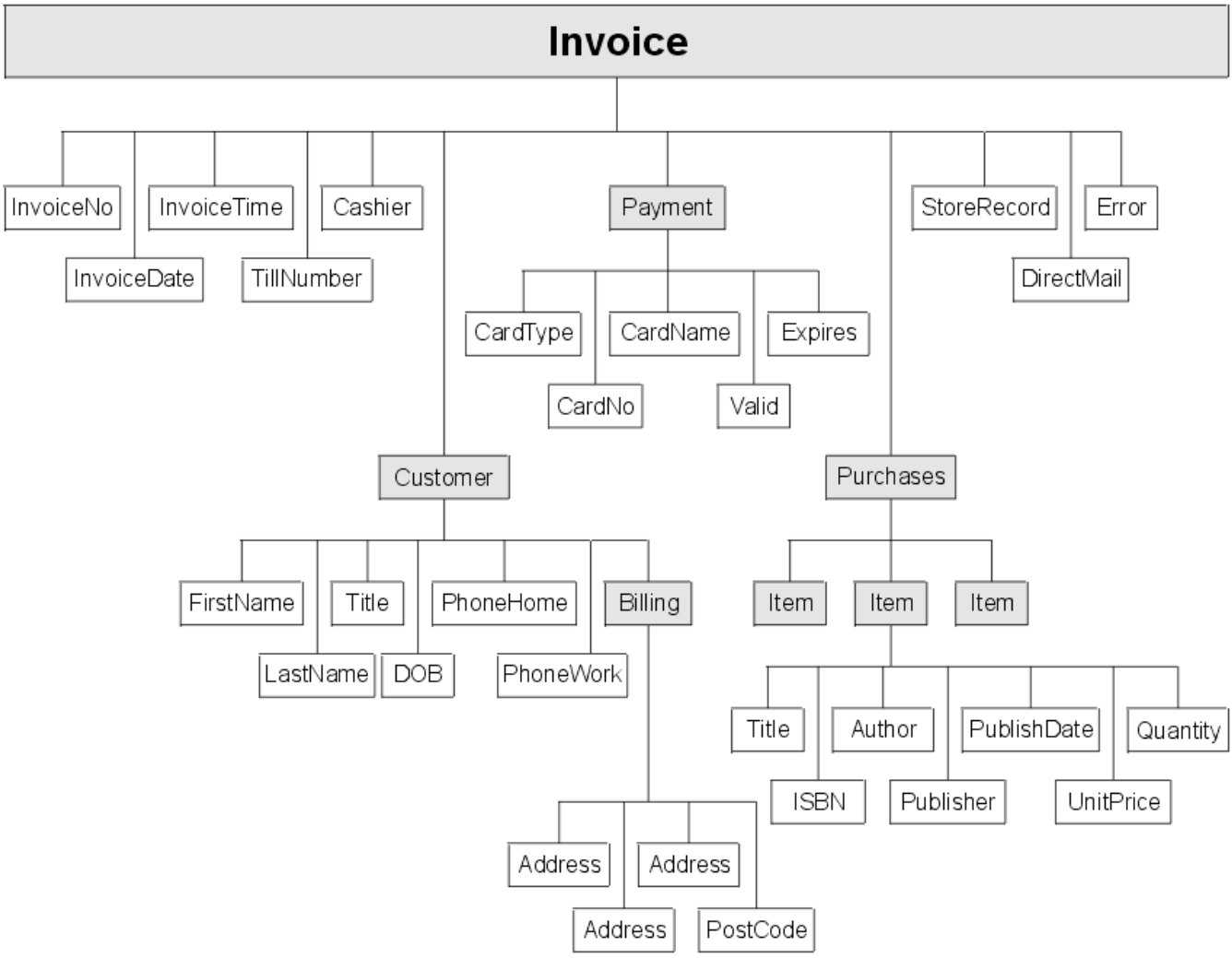
ESQL examples

About this task

Most of the examples included in the topics listed previously show parser-independent ESQL. If examples include a reference to MRM, they assume that you have modeled the message in the MRM and that you have set the names of the MRM objects to be identical to the names of the corresponding tags or attributes in the XML source message. Some examples are also shown for the XML domain. Unless stated otherwise, the principals illustrated are the same for all message domains. For domain-specific information, use the appropriate link in the previous list.

Most of the topics that include example ESQL use the ESQL sample message, Invoice, as the input message to the logic. This message is provided in XML source format (with tags and attributes), see Example message. The example message is shown in the following diagram.

A few other input messages are used to show ESQL that provides function on messages with a structure or content that is not included in the Invoice or Video samples. Where this occurs, the input message is included in the topic that refers to it.



Tailoring ESQL code for different node types
When you code ESQL to configure Compute, Database, DatabaseInput, and Filter node behavior, be aware of the limitations of each type of node.

About this task

Compute node

You can configure the Compute node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Create one or more output messages, with none, some, or all the content of the input message, and propagate these new messages to the next node in the message flow.

  To propagate the input LocalEnvironment to the output LocalEnvironment, remember to set the Compute node property `Compute mode` to an appropriate value. The Environment is always propagated in the output message.

Database node

You can configure the Database node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow.

DatabaseInput node
You can configure the DatabaseInput node to do any of the following operations:

- Respond to updates that are made to the data in a database.
- Create one or more output messages, with none, some, or of all the content retrieved from a database, and propagate these new messages to the next node in the message flow.

Filter node
You can configure the Filter node to do any of the following operations:

- Update data in a database.
- Insert data into a database.
- Delete data from a database.
- Update the environment tree.
- Update the local environment tree.
- Propagate the input message to the next node in the message flow (the terminal through which the message is propagated depends on the result of the filter expression).

View the remaining tasks in this section to find the details of how you can perform these operations.

Manipulating message body content
The message body is always the last child of root, and is identified by its parser name, for example XML or MRM. You can refer to, modify, and create message body data.

About this task

The following topics describe how you can refer to, modify, and create message body data. The information provided here is domain independent.

Referencing field types
Some message parsers have complex models in which it is not enough to identify a field by its name and an array subscript. In these cases, you associate an optional field type with an element of data in the tree format.

About this task

Each element within the parsed tree can be one of three types:

Name element
A name element has a string, which is the name of the element, associated with it. An example of a name element is `XMLElement`, described in XML element.

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

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

Accessing elements in the message body
When you want to access the contents of a message, for reading or writing, use the structure and arrangement of the elements in the tree that is created by the parser from the input bit stream.

About this task

Follow the relevant parent and child relationships from the top of the tree downwards, until you reach the required element.

- If you are referring to the input message tree to interrogate its content in a Compute node, use correlation name `InputBody` followed by the path to the element to which you are referring. `InputBody` is equivalent to `InputRoot` followed by the parser name (for example, `InputRoot.MRM`), which you can use if you prefer.
- If you are referring to the output message tree to set or modify its content in the Compute node, use correlation name `OutputRoot` followed by the parser name (for example, `OutputRoot.MRM`).
- If you are referring to the input message to interrogate its contents in a Database or Filter node, use correlation name `Body` to refer to the start of the message. `Body` is equivalent to `Root` followed by the parser name (for example, `Root.XMLNS`), which you can use if you prefer. You cannot use the `Body` correlation name in a DatabaseInput node.

  You must use these different correlation names because there is only one message to which to refer in a Database or Filter node; you cannot create an output message in these nodes. Use a Compute node to create an output message.

When you construct field references, the names that you use must be valid ESQL identifiers that conform to ESQL rules. If you enclose an item in double quotation marks, ESQL interprets it as an identifier. If you enclose an item in single quotation marks, ESQL interprets it as a character literal. You must enclose all strings (character strings, byte strings, or binary (bit) strings) in quotation marks, as shown in the following examples. To include a single or double quotation mark within a string, include two consecutive single or double quotation marks.

Important: For a full description of field reference syntax, see ESQL field reference overview.

For more information about ESQL data types, see ESQL data types in message flows.

Assume that you have created a message flow that handles the message Invoice, shown in the figure in “Writing ESQL” on page 1629. If, for example, you want to interrogate the element `CardType` from within a Compute node, use the following statement:

```
IF InputBody.Invoice.Payment.CardType='Visa' THEN
   DO;
```

```
     -- more ESQL --
END IF;
```

If you want to make the same test in a Database or Filter node (where the reference is to the single input message), code:

```
IF Body.Invoice.Payment.CardType='Visa' THEN
   DO;
     -- more ESQL --
END IF;
```

If you want to copy an element from an input XML message to an output message in the Compute node without changing it, use the following ESQL:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
              InputBody.Invoice.Customer.FirstName;
```

If you want to copy an element from an input XML message to an output message and update it, for example by folding to uppercase or by calculating a new value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.FirstName =
              UPPER(InputBody.Invoice.Customer.FirstName);
SET OutputRoot.XMLNS.Invoice.InvoiceNo = InputBody.Invoice.InvoiceNo + 1000;
```

If you want to set a STRING element to a constant value, code:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title = 'Mr';
```

You can also use the equivalent statement:

```
SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = 'Mr';
```

If you want to update an INTEGER or DECIMAL, for example the element `TillNumber`, with the value 26, use the following assignment (valid in the Compute node only):

```
SET OutputRoot.MRM.Invoice.TillNumber=26;
```

The integer data type stores numbers using the 64-bit twos complement form, allowing numbers in the range -9223372036854775808 to 9223372036854775807. You can specify hexadecimal notation for integers as well as normal integer literal format. The hexadecimal letters A to F can be written in uppercase or lowercase, as can the X after the initial zero, which is required. The following example produces the same result as the example shown earlier:

```
SET OutputRoot.MRM.Invoice.TillNumber= 0x1A;
```

The following examples show SET statements for element types that do not appear in the Example message.

To set a FLOAT element to a non-integer value, code:

```
SET OutputRoot.MRM.FloatElement1 = 1.2345e2;
```

To set a BINARY element to a constant value, code:

```
SET OutputRoot.MRM.BinaryElement1 = X'F1F1';
```

For BINARY values, you must use an initial character X (uppercase or lowercase) and enclose the hexadecimal characters (also uppercase or lowercase) in single quotation marks, as shown.

To set a BOOLEAN element to a constant value (the value 1 equates to true, 0 equates to false), code:

```
SET OutputRoot.MRM.BooleanElement1 = true;
```

or

```
SET OutputRoot.MRM.BooleanElement1 = 1;
```

You can use the SELECT statement to filter records from an input message without reformatting the records, and without any knowledge of the complete format of each record. Consider the following example:

```
-- Declare local variable
DECLARE CurrentCustomer CHAR 'Smith';

-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I FROM InputRoot.XMLNS.Invoice[] AS I
              WHERE I.Customer.LastName = CurrentCustomer
    );
```

This code writes all records from the input message to the output message if the WHERE condition (LastName = Smith) is met. All records that do not meet the condition are not copied from input message to output message. I is used as an alias for the correlation name InputRoot.XMLNS.Invoice[].

The declared variable CurrentCustomer is initialized on the DECLARE statement: this option is the most efficient way of declaring a variable for which the initial value is known.

You can use this alias technique with other SELECT constructs. For example, if you want to select all the records of the input message, and create an additional record:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I, 'Customer' || I.Customer.LastName AS ExtraField
              FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

You could also include an AS clause to place records in a subfolder in the message tree:

```
-- Loop through the input message
SET OutputRoot.XMLNS.Invoice[] =
    (SELECT I AS Order
            FROM InputRoot.XMLNS.Invoice[] AS I
    );
```

If you are querying or setting elements that contain, or might contain, null values, be aware of the following considerations:

Querying null values

When you compare an element to the ESQL keyword NULL, this tests whether the element is present in the logical tree that has been created from the input message by the parser.

For example, you can check whether an invoice number is included in the current invoice message with the following statement:

```
IF InputRoot.XMLNS.Invoice.InvoiceNo IS NULL THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use an ESQL reference, as shown in the following example:

```
DECLARE cursor REFERENCE TO InputRoot.MRM.InvoiceNo;

IF LASTMOVE(cursor) = FALSE THEN
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does not exist in logical tree';
ELSEIF FIELDVALUE(cursor) IS NULL THEN
   SET OutputRoot.MRM.Analysis =
       'InvoiceNo does exist in logical tree but is defined as an MRM NULL value';
ELSE
   SET OutputRoot.MRM.Analysis = 'InvoiceNo does exist and has a value';
END IF;
```

For more information about declaring and using references, see “Creating dynamic field references” on page 1639. For a description of the LASTMOVE and FIELDVALUE functions, see LASTMOVE function and FIELDTYPE function.

If the message is in the MRM domain, there are additional considerations for querying null elements that depend on the physical format. For further details, see “Querying null values in a message in the MRM domain” on page 1734.

Setting null values

You can use two statements to set null values:

1. If you set the element to NULL by using the following statement, the element is deleted from the message tree:

   ```
   SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
   ```

   If the message is in the MRM domain, there are additional considerations for null values that depend on the physical format. For further details, see “Setting null values in a message in the MRM domain” on page 1735.

   This technique is called implicit null processing.

2. If you set the value of this element to NULL as follows:

   ```
   SET OutputRoot.XMLNS.Invoice.Customer.Title VALUE = NULL;
   ```

   the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element.

   ```
   SET OutputRoot.XMLNS.Invoice.Customer.Title = NULL;
   ```

   If the message is in the MRM domain, the content of the output bit stream depends on the settings of the physical format null handling properties. For further details, see “Setting null values in a message in the MRM domain” on page 1735.

   This technique is called explicit null processing.

If you set an MRM complex element or an XML, XMLNS, or JMS parent element to NULL without using the VALUE keyword, that element and all its children are deleted from the logical tree.

Accessing known multiple occurrences of an element

When you refer to or create the content of messages, it is very likely that the data contains repeating fields. If you know how many instances there are of a repeating field, and you want to access a specific instance of such a field, you can use an array index as part of a field reference.

About this task

For example, you might want to filter on the first line of an address, to expedite the delivery of an order. Three instances of the element Billling.Address are always present in the sample message. To test the first line, write an expression such as:

```
IF Body.Invoice.Customer.Billing.Address[1] = 'Patent Office' THEN
   DO;
     -- more ESQL --
END IF;
```

The array index `[1]` indicates that it is the first instance of the repeating field that you are interested in (array indices start at 1). An array index such as this can be used at any point in a field reference, so you could, for example, filter on the following test:

```
IF Body.Invoice."Item"[1].Quantity> 2 THEN
   DO;
     -- more ESQL --
END IF;
```

You can refer to the last instance of a repeating field using the special [<] array index, and to instances relative to the last (for example, the second to last) as follows:

- `Field[<]` indicates the last element.
- `Field[<1]` indicates the last element.
- `Field[<2]` indicates the last but one element (the penultimate element).

You can also use the array index [>] to represent the first element, and elements relative to the first element in a similar way.

- `Field[>]` indicates the first element. This is equivalent to `Field[1]`.

The following examples refer to the Example message using these indexes:

```
IF Body.Invoice.Customer.Billing.Address[<] = 'Hampshire' THEN
      DO;
     -- more ESQL --
END IF;
IF Body.Invoice.Customer.Billing.Address[<2 ] = 'Southampton' THEN
   DO;
     -- more ESQL --
END IF;
```

You can also use these special indexes for elements that repeat an unknown number of times.

Deleting repeating fields

About this task

If you pass a message with several repeats of an element through a message flow and you want to delete some of the repeats, be aware that the numbering of the repeats is reordered after each delete. For example, if you have a message with five repeats of a particular element, and in the message flow you have the following ESQL:

```
SET OutputRoot.MRM.e_PersonName[1] = NULL;
SET OutputRoot.MRM.e_PersonName[4] = NULL;
```

You might expect elements one and four to be deleted. However, because repeating elements are stored on a stack, when you delete one, the one above it takes its place. This means that, in the above example, elements one and five are deleted. To avoid this problem, delete in reverse order, that is, delete element four first, then delete element one.

Accessing unknown multiple occurrences of an element
To access repeating fields in a message, you must use a construct that can iterate over all instances of a repeating field.

About this task

You are likely to deal with messages that contain repeating fields with an unknown number of repeats (such as the `Item` field in Example message).

To write a filter that takes into account all instances of the `Item` field, you need to use a construct that can iterate over all instances of a repeating field. The quantified predicate allows you to execute a predicate against all instances of a repeating field, and collate the results.

For example, you might want to verify that none of the items that are being ordered has a quantity greater than 50. To do this you could write:

```
FOR ALL Body.Invoice.Purchases."Item"[]
    AS I (I.Quantity <= 50)
```

With the quantified predicate, the first thing to note is the brackets [] on the end of the field reference after FOR ALL. These tell you that you are iterating over all instances of the `Item` field.

In some cases, this syntax appears unnecessary because you can get that information from the context, but it is done for consistency with other pieces of syntax.

The AS clause associates the name `I` with the current instance of the repeating field. This is similar to the concept of iterator classes used in some object oriented languages such as C++. The expression in parentheses is a predicate that is evaluated for each instance of the `Item` field.

A description of this example is:

Procedure

Iterate over all instances of the field Item inside `Body.Invoice`.

For each iteration:

a) Bind the name `I` to the current instance of `Item`.

b) Evaluate the predicate `I.Quantity <= 50`.

If the predicate:

- Evaluates to TRUE for all instances of `Item`, return TRUE.
- Is FALSE for any instance of `Item`, return FALSE.
- For a mixture of TRUE and UNKNOWN, return UNKNOWN.

Results

The above is a description of how the predicate is evaluated if you use the ALL keyword. An alternative is to specify SOME, or ANY, which are equivalent. In this case the quantified predicate returns TRUE if the sub-predicate returns TRUE for any instance of the repeating field. Only if the sub-predicate returns FALSE for all instances of the repeating field does the quantified predicate return FALSE. If a mixture of FALSE and UNKNOWN values are returned from the sub-predicate, an overall value of UNKNOWN is returned.

In the following filter expression:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'The XML Companion')
```

the sub-predicate evaluates to TRUE. However this next expression returns FALSE:

```
FOR ANY Body.Invoice.Purchases."Item"[]
    AS I (I.Title = 'C Primer')
```

because the `C Primer` is not included on this invoice. If some of the items in the invoice do not include a book title field, the sub-predicate returns UNKNOWN, and the quantified predicate returns the value UNKNOWN.

To deal with the possibility of null values appearing, write this filter with an explicit check on the existence of the field, as follows:

```
FOR ANY Body.Invoice.Purchases."Item"[]
  AS I (I.Book IS NOT NULL AND I.Book.Title = 'C Primer')
```

The predicate IS NOT NULL ensures that, if an `Item` field does not contain a Book, a FALSE value is returned from the sub-predicate.

You can also manipulate arbitrary repeats of fields within a message by using a SELECT expression, as described in “Referencing columns in a database” on page 1666.

You can refer to the first and last instances of a repeating field using the [>] and [<] array indexes, and to instances relative to the first and last, even if you do not know how many instances there are. These indexes are described in “Accessing known multiple occurrences of an element” on page 1636.

Alternatively, you can use the CARDINALITY function to determine how many instances of a repeating field there are. For example:

```
DECLARE I INTEGER CARDINALITY(Body.Invoice.Purchases."Item"[])
```

Using anonymous field references
You can refer to the array of all children of a particular element by using a path element of *.

About this task

For example:

```
InputRoot.*[]
```

is a path that identifies the array of all children of InputRoot. This is often used in conjunction with an array subscript to refer to a particular child of an entity by position, rather than by name. For example:

InputRoot.*[<]
Refers to the last child of the root of the input message, that is, the body of the message.

InputRoot.*[1]
Refers to the first child of the root of the input message, the message properties.

You might want to find out the name of an element that has been identified with a path of this kind. To do this, use the FIELDNAME function, which is described in FIELDNAME function.

Creating dynamic field references
You can use a variable of type REFERENCE as a dynamic reference to navigate a message tree. This acts in a similar way to a message cursor or a variable pointer.

About this task

It is generally simpler and more efficient to use reference variables in preference to array indexes when you access repeating structures. Reference variables are accepted everywhere that field references are accepted, and come with a set of statements and functions to allow detailed manipulation of message trees.

You must declare a dynamic reference before you can use it. A dynamic reference is declared and initialized in a single statement.

All examples in this topic use the Example message as their input message. The following example shows how to create and use a reference.

```
-- Declare the dynamic reference
DECLARE myref REFERENCE TO OutputRoot.XMLNS.Invoice.Purchases.Item[1];

--  Continue processing for each item in the array
WHILE LASTMOVE(myref)=TRUE
DO
-- Add 1 to each item in the array
   SET myref = myref + 1;
-- Move the dynamic reference to the next item in the array
   MOVE myref NEXTSIBLING;
END WHILE;
```

The example works if the message tree was created with typed fields, based on a message model for the Example message. If this is not the case, you can modify the ESQL to work without a model, for example:

```
SET myref = CAST (myref AS INTEGER) + 1;
```

This example declares a dynamic reference, `myref`, which points to the first item in the array within Purchases. The value in the first item is incremented by one, and the pointer (dynamic reference) is moved to the next item. Once again the item value is incremented by one. This process continues until the pointer moves outside the scope of the message array (all the items in this array have been processed) and the LASTMOVE function returns FALSE.

The following examples show further examples.

```
DECLARE ref1 REFERENCE TO InputBody.Invoice.Purchases.Item[1];

DECLARE ref2 REFERENCE TO
 InputBody.Invoice.Purchases.NonExistentField;

DECLARE scalar1 CHARACTER;
DECLARE ref3 REFERENCE TO scalar1;
```

In the second example, `ref2` is set to point to InputBody because the specified field does not exist.

With the exception of the MOVE statement, which changes the position of the dynamic reference, you can use a dynamic reference anywhere that you can use a static reference. The value of the dynamic reference in any expression or statement is the value of the field or variable to which it currently points. For example, using the message in Example message, the value of `Invoice.Customer.FirstName` is Andrew. If the dynamic reference `myref` is set to point at the Customer field as follows:

```
DECLARE myref REFERENCE TO Invoice.Customer;
```

you can then extend this dynamic reference to address children of that field:

```
SET myref.Billing.Address[1] = 'Oaklands';
```

This changes the address in the example to Oaklands Hursley Village Hampshire SO213JR.

The position of a dynamic reference remains fixed even if a tree is modified. To illustrate this point the steps that follow use the message in Example message as their input message and create a modified version of this message as an output message:

Procedure

1. Copy the input message to the output message.
2. To modify the output message, first declare a dynamic reference `ref1` that points at the first item, The XML Companion.

   ```
   DECLARE ref1 REFERENCE TO
    OutputRoot.XMLNS.Invoice.Purchases.Item[1];
   ```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[1].`
3. Use a create statement to insert a new first item for this purchase.

   ```
   CREATE PREVIOUSSIBLING OF ref1 VALUE 'Item';
   ```

   The dynamic reference is now equivalent to the static reference `OutputRoot.XMLNS.Invoice.Purchases.Item[2].`

Creating new fields
You can use a Compute node to create a new output message by adding new fields to an existing input message.

About this task

This topic provides example ESQL code for a Compute node that creates a new output message based on the input message, to which are added a number of additional fields.

The input message received by the Compute node within the message flow is an XML message, and has the following content:

```
<TestCase description="This is my TestCase">
    <Identifier>ES03B305_T1</Identifier>
    <Sport>Football</Sport>
    <Date>01/02/2000</Date>
    <Type>LEAGUE</Type>
</TestCase>
```

The Compute node is configured and an ESQL module is created that includes the following ESQL. The following code copies the headers from the input message to the new output message, then creates the entire content of the output message body.

```
-- copy headers
DECLARE i INTEGER 1;
DECLARE numHeaders INTEGER CARDINALITY(InputRoot.*[]);

WHILE i < numHeaders DO
    SET OutputRoot.*[i] = InputRoot.*[i];
    SET i = i + 1;
END WHILE;

CREATE FIELD OutputRoot.XMLNS.TestCase.description TYPE NameValue VALUE 'This is my TestCase';
CREATE FIRSTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Identifier'
                               VALUE InputRoot.XMLNS.TestCase.Identifier;
CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Sport'
                               VALUE InputRoot.XMLNS.TestCase.Sport;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Date'
                               VALUE InputRoot.XMLNS.TestCase.Date;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase Domain('XMLNS') NAME 'Type'
                               VALUE InputRoot.XMLNS.TestCase.Type;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Number TYPE NameValue
                               VALUE 'Premiership';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Number TYPE NameValue VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Team'
                               VALUE 'Liverpool' ;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Home NAME 'Score'
                               VALUE '4';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Team'
                               VALUE 'Everton';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[1].Away NAME 'Score'
                               VALUE '0';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Number TYPE NameValue VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Team'
                               VALUE 'Manchester United';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Home NAME 'Score'
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Team'
                               VALUE 'Arsenal';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[1].Result[2].Away NAME 'Score'
                               VALUE '3';

 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Number TYPE NameValue
                               VALUE '2';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Number TYPE NameValue
                               VALUE  '1';
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Team'
                               VALUE 'Port Vale';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Home NAME 'Score'
                               VALUE '9' ;
 CREATE FIELD OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away TYPE Name;
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Team'
                               VALUE 'Brentford';
 CREATE LASTCHILD OF OutputRoot.XMLNS.TestCase.Division[2].Result[1].Away NAME 'Score'
                               VALUE '5';
```

The output message that results from the ESQL shown above has the following structure and content:

```
<TestCase description="This is my TestCase">
  <Identifier>ES03B305_T1</Identifier>
  <Sport>Football</Sport>
  <Date>01/02/2000</Date>
  <Type>LEAGUE</Type>
  <Division Number="Premiership">
    <Result Number="1">
      <Home>
        <Team>Liverpool</Team>
        <Score>4</Score>
      </Home>
      <Away>
        <Team>Everton</Team>
        <Score>0</Score>
```

```
      </Away>
    </Result>
    <Result Number="2">
      <Home>
        <Team>Manchester United</Team>
        <Score>2</Score>
      </Home>
      <Away>
        <Team>Arsenal</Team>
        <Score>3</Score>
      </Away>
    </Result>
  </Division>
  <Division Number="2">
    <Result Number="1">
      <Home>
        <Team>Port Vale</Team>
        <Score>9</Score>
      </Home>
      <Away>
        <Team>Brentford</Team>
        <Score>5</Score>
      </Away>
    </Result>
  </Division>
</TestCase>
```

Generating multiple output messages

You can use the PROPAGATE statement to generate multiple output messages in the Compute node. The output messages that you generate can have the same or different content. You can also direct output messages to any of the four alternate output terminals of the Compute node, or to a Label node.

About this task

For example, to create three copies of the input message received by the Compute node, and send one to the standard Out terminal of the Compute node, one to the first alternate Out1 terminal of the Compute node, and one to the Label node ThirdCopy, code the following ESQL:

```
SET OutputRoot = InputRoot;
PROPAGATE;
SET OutputRoot = InputRoot;
PROPAGATE TO TERMINAL 'out1';
SET OutputRoot = InputRoot;
PROPAGATE TO LABEL 'ThirdCopy';
```

In the above example, the content of OutputRoot is reset before each PROPAGATE, because by default the node clears the output message buffer and reclaims the memory when the PROPAGATE statement completes. An alternative method is to instruct the node not to clear the output message on the first two PROPAGATE statements, so that the message is available for routing to the next destination. The code to achieve this result is:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE TO TERMINAL 'out1' DELETE NONE;
PROPAGATE TO LABEL 'ThirdCopy';
```

If you do not initialize the output buffer, an empty message is generated, and the message flow detects an error and throws an exception.

Also ensure that you copy all required message headers to the output message buffer for each output message that you propagate.

If you want to modify the output message content before propagating each message, code the appropriate ESQL to make the changes that you want before you code the PROPAGATE statement.

If you set up the contents of the last output message that you want to generate, and propagate it as the final action of the Compute node, you do not have to include the final PROPAGATE statement. The default action of the Compute node is to propagate the contents of the output buffer when it terminates. This is implemented by the RETURN TRUE statement, included as the final statement in the module skeleton.

For example, to generate three copies of the input message, and not perform any further action, include this code immediately before the RETURN TRUE statement:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
```

Alternatively, you can modify the default behavior of the node by changing RETURN TRUE to RETURN FALSE:

```
SET OutputRoot = InputRoot;
PROPAGATE DELETE NONE;
PROPAGATE DELETE NONE;
PROPAGATE;
RETURN FALSE;
```

Three output messages are generated by the three PROPAGATE statements. The final RETURN FALSE statement causes the node to terminate but not propagate a final output message. Note that the final PROPAGATE statement does not include the DELETE NONE clause, because the node must release the memory at this stage.

Using numeric operators with datetime values
The following examples show the ESQL that you can code to manipulate datetime values with numeric operators.

About this task

Adding an interval to a datetime value

The simplest operation that you can perform is to add an interval to, or subtract an interval from, a datetime value. For example, you could write the following expressions:

```
DATE '2000-03-29' + INTERVAL '1' MONTH
TIMESTAMP '1999-12-31 23:59:59' + INTERVAL '1' SECOND
```

The following example shows how to calculate a retirement date by adding the retirement age to the birth date.

```
DECLARE retAge CHARACTER '65';
DECLARE birthDate DATE DATE '1953-06-01';

SET OutputRoot.XML.Test.retirementDate = birthDate + CAST(retAge AS INTERVAL YEAR);
```

The repetition of the word DATE in the above example is intentional. The first occurrence of DATE specifies the data type of the declared variable, birthDate. The second occurrence initializes the same variable with the constant character string that is enclosed in single quotation marks as a DATE.

Adding or subtracting two intervals

You can combine two interval values by using addition or subtraction. The two interval values must be of compatible types. It is not valid to add a year-month interval to a day-second interval as shown in the following example:

```
INTERVAL '1-06' YEAR TO MONTH + INTERVAL '20' DAY
```

The interval qualifier of the resultant interval is sufficient to encompass all the fields that are present in the two operand intervals. For example:

```
INTERVAL '2 01' DAY TO HOUR + INTERVAL '123:59' MINUTE TO SECOND
```

results in an interval with qualifier DAY TO SECOND, because both day and second fields are present in at least one of the operand values.

Subtracting two datetime values

You can subtract two datetime values to return an interval. You must include an interval qualifier in the expression to indicate what precision the result should be returned in. For example:

```
(CURRENT_DATE - DATE '1776-07-04') DAY
```

returns the number of days since the 4th July 1776, whereas:

```
(CURRENT_TIME - TIME '00:00:00') MINUTE TO SECOND
```

returns the age of the day in minutes and seconds.

Scaling intervals

You can multiply or divide an interval value by an integer factor:

```
INTERVAL '2:30' MINUTE TO SECOND / 4
```

Calculating a time interval

You can use ESQL to calculate the time interval between two events, and to set a timer to be triggered after a specified interval.

About this task

This ESQL example calculates the time interval between an input IBM MQ message being put on the input queue, and the time that it is processed in the current Compute node.

(When you make a call to a CURRENT_ datetime function, the value that is returned is identical to the value returned to another call in the same node. This ensures that you can use the function consistently within a single node.)

```
 CALL CopyMessageHeaders();
 Declare PutTime INTERVAL;

 SET PutTime = (CURRENT_GMTTIME - InputRoot.MQMD.PutTime) MINUTE TO SECOND;

 SET OutputRoot.XMLNS.Test.PutTime = PutTime;
```

The output message has the format (although actual values vary):

```
<Test>
 <PutTime>INTERVAL &apos;1:21.862&apos; MINUTE TO SECOND</PutTime>
</Test>
```

Example

The following code snippet sets a timer, to be triggered after a specified interval from the start of processing, in order to check that processing has completed. If processing has not completed within the elapsed time, the firing of the timer might, for example, trigger some recovery processing.

The `StartTime` field of the timeout request message is set to the current time plus the allowed delay period, which is defined by a user-defined property on the flow. (The user-defined property has been set to a string of the form "HH:MM:SS" by the administrator.)

```
DECLARE StartDelyIntervalStr EXTERNAL CHARACTER '01:15:05';

 CREATE PROCEDURE ValidateTimeoutRequest() BEGIN

  -- Set the timeout period
  DECLARE timeoutStartTimeRef REFERENCE TO
         OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime;
  IF LASTMOVE(timeoutStartTimeRef)
   THEN
   -- Already set
  ELSE
   -- Set it from the UDP StartDelyIntervalStr
   DECLARE startAtTime TIME CURRENT_TIME
          + CAST(StartDelyIntervalStr AS INTERVAL HOUR TO SECOND);

   -- Convert "TIME 'hh.mm.ss.fff'" to hh.mm.ss format
       -- needed in StartTime field
   DECLARE startAtTimeStr CHAR;
   SET startAtTimeStr = startAtTime;
```

```
   SET startAtTimeStr = SUBSTRING(startAtTimeStr FROM 7 FOR  8);
   SET OutputRoot.XMLNSC.Envelope.Header.TimeoutRequest.StartTime
                            = startAtTimeStr;
  END IF;
 END;
```

Selecting a subfield from a larger field
You might have a message flow that processes a message containing delimited subfields. You can code ESQL to extract a subfield from the surrounding content if you know the delimiters of the subfield.

About this task

If you create a function that performs this task, or a similar one, you can invoke it both from ESQL modules (for Compute, Database, DatabaseInput, and Filter nodes) and from mapping files (used by the Mapping node).

The following function example extracts a particular subfield of a message that is delimited by a specific character.

```
CREATE FUNCTION SelectSubField
       (SourceString CHAR, Delimiter CHAR, TargetStringPosition INT)
d       RETURNS CHAR
-- This function returns a substring at parameter position TargetStringPosition within the
-- passed parameter SourceString.  An example of use might be:
-- SelectSubField(MySourceField,' ',2) which will select the second subfield from the
-- field MySourceField delimited by a blank.  If MySourceField has the value
-- "First Second Third" the function will return the value "Second"
  BEGIN
    DECLARE DelimiterPosition INT;
    DECLARE CurrentFieldPosition INT 1;
    DECLARE StartNewString INT 1;
    DECLARE WorkingSource CHAR SourceString;
    SET DelimiterPosition = POSITION(Delimiter IN SourceString);
    WHILE CurrentFieldPosition < TargetStringPosition
      DO
       IF DelimiterPosition = 0  THEN
      -- DelimiterPosition will be 0 if the delimiter is not found
         -- exit the loop
        SET CurrentFieldPosition = TargetStringPosition;
       ELSE
        SET StartNewString = DelimiterPosition + 1;
        SET WorkingSource = SUBSTRING(WorkingSource FROM StartNewString);
        SET DelimiterPosition = POSITION(Delimiter IN WorkingSource);
        SET CurrentFieldPosition = CurrentFieldPosition + 1;
       END IF;
    END WHILE;
    IF DelimiterPosition> 0 THEN
       -- Remove anything following the delimiter from the string
       SET WorkingSource = SUBSTRING(WorkingSource FROM 1 FOR DelimiterPosition);
       SET WorkingSource = TRIM(TRAILING Delimiter FROM WorkingSource);
    END  IF;
    RETURN WorkingSource;
END;
```

Copying repeating fields
You can configure a node with ESQL to copy repeating fields in several ways.

About this task

Consider an input XML message that contains a repeating structure:

```
...
 <Field_top>
   <field1></field1>
   <field1></field1>
   <field1></field1>
   <field1></field1>
   <field1></field1>
 </Field_top>
.....
```

You cannot copy this whole structure field with the following statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1 = InputRoot.XMLNS.Field_top.field1;
```

That statement copies only the first repeat, and therefore produces the same result as this statement:

```
SET OutputRoot.XMLNS.Output_top.Outfield1[1] = InputRoot.XMLNS.Field_top.field1[1];
```

You can copy the fields within a loop, controlling the iterations with the CARDINALITY of the input field:

```
SET I = 1;
SET J = CARDINALITY(InputRoot.XMLNS.Field_top.field1[]);
WHILE  I <= J DO
 SET OutputRoot.XMLNS.Output_top.Outfield1[I] = InputRoot.XMLNS.Field_top.field1[I];
 SET I = I + 1;
END WHILE;
```

This might be appropriate if you want to modify each field in the output message as you copy it from the input field (for example, add a number to it, or fold its contents to uppercase), or after it has been copied. If the output message already contained more Field1 fields than existed in the input message, the surplus fields would not be modified by the loop and would remain in the output message.

The following single statement copies the iterations of the input fields to the output fields, and deletes any surplus fields in the output message.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.[] = InputRoot.XMLNS.Field_top.field1[];
```

The following example shows how you can rename the elements when you copy them into the output tree. This statement does not copy across the source element name, therefore each `field1` element becomes a `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield1.Target[] =
    (SELECT I FROM InputRoot.XMLNS.Field_top.field1[] AS I );
```

The next example shows a different way to do the same operation; it produces the same end result.

```
SET OutputRoot.XMLNS.Output_top.Outfield2.Target[]
             = InputRoot.XMLNS.Field_top.field1[];
```

The following example copies across the source element name. Each `field1` element is retained as a `field1` element under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield3.Target.[]
             = InputRoot.XMLNS.Field_top.field1[];
```

This example is an alternative way to achieve the same result, with `field1` elements created under the `Target` element.

```
SET OutputRoot.XMLNS.Output_top.Outfield4.Target.*[]
             = InputRoot.XMLNS.Field_top.field1[];
```

These examples show that there are several ways in which you can code ESQL to copy repeating fields from source to target. Select the most appropriate method to achieve the results that you require.

The principals shown here apply equally to all areas of the message tree to which you can write data, not just the output message tree.

Working with elements of type list
The XML Schema specification allows an element, or attribute, to contain a list of values that are based on a simple type, with the individual values separated by white space.

About this task

In the message tree, an `xsd:list` element is represented as a name node, with an anonymous child node for each list item. Repeating lists can be handled without any loss of information.

Consider the following XML input message:

```
<message1>
  <listE1>one two three</listE1>
</message1>
```

This XML element produces the following message tree:

```
MRM
listEl  (Name)
 "one"   (Value)
 "two"   (Value)
 "three" (Value)
```

Individual list items can be accessed as `ElementName.*[n]`.

For example, use the following ESQL to access the third item of listE1:

```
SET X = FIELDVALUE (InputBody.message1.listE1.*[3]);
```

An attribute can also be of type `xsd:list`.

Consider the following XML input message:

```
<message1>
<Element listAttr="one two three"/>
</message1>
```

This XML element produces the following message tree:

```
MRM
Element (Name)
 listAttr (Name)
  "one" (Value)
  "two" (Value)
  "three" (Value)
```

As before, individual list items can be accessed as `AttrName.*[n]`.

For example, use the following ESQL to access the third item of listAttr:

```
SET X = FIELDVALUE (InputBody.message1.Element.listAttr.*[3]);
```

A list element can occur more than once.

Consider the following XML message:

```
<message1>
<listE1>one two three/listE1>
<listE1>four five six/listE1>
</message1>
```

The message tree for this XML message is:

```
MRM
listE1 (Name)
 "one" (Value)
 "two" (Value)
 "three" (Value)
listE1 (Name)
 "four" (Value)
```

```
  "five" (Value)
  "six" (Value)
```

Code the following ESQL to access the first item in the second occurrence of the list:

```
SET X = FIELDVALUE (InputBody.message1.listE1[2].*[1]);
```

Copying data between a list and a repeating element
This task shows how to copy data from a list into a repeating element by using ESQL.

About this task

Consider the form of the following XML input message:

```
<MRM>
  <inner>abcde fghij 12345</inner>
</MRM>
```

where the element inner is of type `xsd:list`, and therefore has three associated string values, rather than a single value.

To copy the three values into an output message, where each value is associated with an instance of repeating elements as shown here:

```
<MRM>
  <str1>abcde</str1>
  <str1>fghij</str1>
  <str1>12345</str1>
</MRM>
```

you might expect that the following ESQL syntax works:

```
DECLARE D INTEGER;
SET D = CARDINALITY(InputBody.str1.*[]);
DECLARE M INTEGER 1;
WHILE M <= D DO
   SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
   SET M = M + 1;
END WHILE;
```

However, the statement:

```
SET OutputRoot.MRM.str1[M] = InputBody.inner.*[M];
```

requests a tree copy from input to output. Because the output element does not yet exist, the statement creates it, and its value and type are set from the input.

Therefore, to create the output message with the required format, given an input element which is of type `xsd:list`, use the FIELDVALUE function to explicitly retrieve only the value of the input element:

```
SET OutputRoot.MRM.str1[M] = FIELDVALUE(InputBody.inner.*[M]);
```

Manipulating repeating fields in a message tree
This topic describes the use of the SELECT function, and other column functions, to manipulate repeating fields in a message tree.

About this task

Suppose that you want to perform a special action on invoices that have a total order value greater than a certain amount. To calculate the total order value of an `Invoice` field, you must multiply the `Price` fields by the `Quantity` fields in all the `Items` in the message, and total the result. You can do this using a SELECT expression as follows:

```
(
 SELECT SUM( CAST(I.Price AS DECIMAL) * CAST(I.Quantity AS INTEGER) )
  FROM Body.Invoice.Purchases."Item"[] AS I
)
```

The example assumes that you need to use CAST expressions to cast the string values of the fields `Price` and `Quantity` into the correct data types. The cast of the `Price` field into a decimal produces a decimal value with the *natural* scale and precision, that is, whatever scale and precision is necessary to represent the number. These CASTs would not be necessary if the data were already in an appropriate data type.

The SELECT expression works in a similar way to the quantified predicate, and in much the same way that a SELECT works in standard database SQL. The FROM clause specifies what is being iterated, in this case, all `Item` fields in `Invoice`, and establishes that the current instance of `Item` can be referred to using `I`. This form of SELECT involves a column function, in this case the SUM function, so the SELECT is evaluated by adding together the results of evaluating the expression inside the SUM function for each `Item` field in the `Invoice`. As with standard SQL, NULL values are ignored by column functions, with the exception of the COUNT column function explained later in this section, and a NULL value is returned by the column function only if there are no non-NULL values to combine.

The other column functions that are provided are MAX, MIN, and COUNT. The COUNT function has two forms that work in different ways with regard to NULLs. In the first form you use it much like the SUM function above, for example:

```
SELECT COUNT(I.Quantity)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

This expression returns the number of `Item` fields for which the `Quantity` field is non-NULL. That is, the COUNT function counts non-NULL values, in the same way that the SUM function adds non-NULL values. The alternative way of using the COUNT function is as follows:

```
SELECT COUNT(*)
  FROM Body.Invoice.Purchases."Item"[] AS I
```

Using COUNT(*) counts the total number of `Item` fields, regardless of whether any of the fields is NULL. The preceding example is in fact equivalent to using the CARDINALITY function, as in the following example:

```
CARDINALITY(Body.Invoice.Purchases."Item"[])
```

In all the examples of SELECT given here, just as in standard SQL, you could use a WHERE clause to provide filtering on the fields.

Manipulating other parts of the message tree
You can access message tree headers, the properties tree, the local environment tree, the environment tree and the exception list tree.

About this task

The following topics describe how you can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in the BLOB domain in addition to all other supported domains.

- “Accessing headers” on page 1650
- “Accessing the Properties tree” on page 1653
- “Accessing the local environment tree” on page 1654
- “Accessing the environment tree” on page 1657
- “Accessing the ExceptionList tree using ESQL” on page 1658

Accessing headers
If the input message received by an input node includes message headers that are recognized by the input node, the node invokes the correct parser for each header. You can access these headers using ESQL.

About this task

Parsers are supplied for most IBM MQ headers. The following topics provide some guidance for accessing the information in the MQMD, MQRFH2, and MQPCF headers, which you can follow as general guidance for accessing other headers also present in your messages.

Every header has its own correlation name, for example, MQMD, and you must use this name in all ESQL statements that refer to or set the content of this tree:

- "Accessing the MQMD header" on page 1650
- "Accessing the MQRFH2 header" on page 1651
- "Accessing the MQCFH header" on page 1651

For further details of the contents of these and other IBM MQ headers for which IBM App Connect Enterprise provides a parser, see Element definitions for message parsers.

Accessing transport headers

About this task

You can manipulate IBM MQ, HTTP, and JMS transport headers and their properties without writing Compute nodes:

- Use the MQHeader node to add, modify, or delete MQ Message Descriptor (MQMD) and MQ Dead Letter Header (MQDLH) headers.
- Use the HTTPHeader node to add, modify, or delete HTTP headers such as HTTPRequest and HTTPReply.
- Use the JMSHeader node to modify contents of the JMS Header_Values and Application properties so that you can control the node's output without programming.

Accessing the MQMD header
Code ESQL statements to access the fields of the MQMD header.

About this task

IBM MQ and WebSphere MQ Everyplace® messages include an MQMD header.

You can refer to the fields within the MQMD, and you can update them in a Compute node.

For example, you might want to copy the message identifier MSGID in the MQMD to another field in your output message. To do that, code:

```
SET OutputRoot.MRM.Identifier = InputRoot.MQMD.MsgId;
```

If you send a message to an EBCDIC system from a distributed system, you might need to convert the message to a compatible CodedCharSetId and Encoding. To do this conversion, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
```

The MQMD properties of CodedCharSetId and Encoding define the code page and encoding of the section of the message that follows (typically this is either the MQRFH2 header or the message body itself).

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see "Properties versus MQMD folder behavior for various transports" on page 506.

Accessing the MQRFH2 header
Code ESQL statements to access the fields of the MQRFH2 header.

About this task

When you construct an MQRFH2 header in a Compute node, it includes two types of fields:

- Fields in the MQRFH2 header structure; for example, Format and NameValueCCSID.
- Fields in the MQRFH2 NameValue buffer; for example, mcd and psc.

To differentiate between these two field types, insert a value in front of the referenced field in the MQRFH2 field to identify its type; a value for the NameValue buffer is not required because this is the default. The value that you specify for the header structure is `(MQRFH2.Field)`.

For example:

- To create or change an MQRFH2 Format field, specify the following ESQL:

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR   ';
```

- To create or change the psc folder with a topic:

```
SET OutputRoot.MQRFH2.psc.Topic = 'department';
```

- To add an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
 SET OutputRoot.*[I] = InputRoot.*[I];
 SET I=I+1;
END WHILE;

SET OutputRoot.MQRFH2.(MQRFH2.Field)Version = 2;
SET OutputRoot.MQRFH2.(MQRFH2.Field)Format = 'MQSTR';
SET OutputRoot.MQRFH2.(MQRFH2.Field)NameValueCCSID = 1208;
SET OutputRoot.MQRFH2.psc.Topic = "InputRoot"."MRM"."topel";
SET OutputRoot.MQRFH2.psc.QMgrName = 'DebugQM';
SET OutputRoot.MQRFH2.psc.QName = 'PUBOUT';
SET OutputRoot.MQRFH2.psc.RegOpt = 'PersAsPub';
```

  Variable J is initialized to the value of the cardinality of the existing headers in the message. Using a variable is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

The MQRFH2 header can be parsed using either the MQRFH2 parser or the MQRFH2C compact parser. To consume less memory, use the MQRFH2C compact parser by selecting the `Use MQRFH2C compact parser for MQRFH2 Header` check box on the input node of the message flow. This results in paths that contain MQRFH2C instead of MQRFH2; for example: `SET OutputRoot.MQRFH2C.psc.Topic = 'department';`

Target MQRFH2 fields are created only if the headers are copied, and the MQRFH2C parser option is not selected on the MQInput node. In all other circumstances, an MQRFH2C field is created on output.

Accessing the MQCFH header
Code ESQL statements to access the fields of the MQCFH header (root name MQPCF).

About this task

Messages that are of PCF format (MQPCF, MQADMIN, and MQEVENT) include the MQCFH header. You can process the contents of the MQCFH header, accessing parameters, parameter lists, strings, and groups.

The *ParameterCount* field is hidden from view to keep the value synchronized with the true number of parameters. As a result, you cannot directly view, access, or edit the *ParameterCount* field and this applies when you are using:

- ESQL
- The Mapping Node
- Trace, or debugging code.

You can implement your own *ParameterCount* field with a specific value, but this value will be overwritten by the actual number of parameters on the flow exit.

- To access or to construct parameters in the header, use the following syntax:

```
SET OutputRoot.MQPCF.Parameter[nn] =
 Integer parameter ID
```

  If you access a 64-bit parameter, use the following syntax to differentiate from 32-bit parameters:

```
SET OutputRoot.MQPCF.Parameter64[nn] =
 Integer parameter ID
```

  For example:

```
SET OutputRoot.MQPCF.Parameter[1] =
 MQCACF_AUTH_PROFILE_NAME;
```

- For parameter lists, use the following syntax:

```
SET OutputRoot.MQPCF.ParameterList64[nn] =
 Integer parameter ID
SET OutputRoot.MQPCF.ParameterList64[nn].*[xx] =
 Integer parameter values
```

  For example:

```
SET OutputRoot.MQPCF.ParameterList[1] =
 MQIACF_AUTH_ADD_AUTHS;
SET OutputRoot.MQPCF.ParameterList[1].*[1] =
 MQAUTH_SET;
SET OutputRoot.MQPCF.ParameterList[1].*[2] =
 MQAUTH_SET_ALL_CONTEXT;
```

- A byte string is a byte array data type, and is supported with this syntax:

```
SET OutputRoot.MQPCF.Parameter[nn] =
 Integer parameter ID
SET OutputRoot.MQPCF.Parameter[nn].* =
 Integer, String or ByteArray Parameter value
```

- A group is implemented as a folder containing more PCF, and requires the following syntax:

```
SET OutputRoot.MQPCF.Group[xx] =
 Group Parameter ID
```

  For example:

```
SET OutputRoot.MQPCF.Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Parameter[1].* =
 '2007-02-05';
```

  You can also use nested groups; for example;

```
SET OutputRoot.MQPCF.Group[1].Group[1] =
 MQGACF_Q_ACCOUNTING_DATA;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1] =
 MQCA_CREATION_DATE;
SET OutputRoot.MQPCF.Group[1].Group[1].Parameter[1].* =
```

```
 '2007-02-05';
```

- A filter is almost identical to a parameter, but it also includes an operator. Therefore the syntax is a tree with named values:

```
SET OutputRoot.MQPCF.Filter[xx] =
 Integer parameter ID
SET OutputRoot.MQPCF.Filter[xx].Operator =
 Integer Filter name
SET OutputRoot.MQPCF.Filter[xx].Value =
 Byte, Integer or String Filter Value
```

- The following is sample code that can be used as an example to create an MQPCF message in a Compute node:

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
CREATE NEXTSIBLING OF OutputRoot.MQMD DOMAIN 'MQADMIN'
NAME 'MQPCF';
CREATE FIELD OutputRoot.MQPCF;
SET OutputRoot.MQMD.MsgType = MQMT_REQUEST;
SET OutputRoot.MQMD.ReplyToQ = 'REPLYQ';
DECLARE refRequest REFERENCE TO OutputRoot.MQPCF;
SET refRequest.Type = 16; --MQCFT_COMMAND_XR z/OS
SET refRequest.StrucLength = MQCFH_STRUC_LENGTH;
SET refRequest.Version = 3; -- required for z/OS
SET refRequest.Command = MQCMD_INQUIRE_Q;
SET refRequest.MsgSeqNumber = 1;
SET refRequest.Control = MQCFC_LAST;
/* First parameter: Queue Name. */
SET refRequest.Parameter[1] = MQCA_Q_NAME;
SET refRequest.Parameter[1].* = 'QUEUENAME.*';
SET refRequest.ParameterList[1] = MQIACF_Q_ATTRS;
SET refRequest.ParameterList[1].*  = MQIACF_ALL;
```

Accessing the Properties tree
The Properties tree has its own correlation name, Properties, and you must use this in all ESQL statements that refer to or set the content of this tree.

About this task

The fields in the Properties tree contain values that define the characteristics of the message. For example, the Properties tree contains message template information for model-driven parsers, fields for the encoding and CCSID in which message data is encoded, and fields that hold the security identity of the message. For a full list of fields in this tree, see Data types for elements in the Properties subtree.

You can interrogate and update these fields using the appropriate ESQL statements. If you create a new output message in the Compute node, you must set values for the message properties.

Setting output message properties

About this task

If you use the Compute node to generate a new output message, you must set its properties in the Properties tree. The output message properties do not have to be the same as the input message properties.

For example, to set the output message properties for an output MRM message, set the following properties:

Property	Value
MessageSet	Message set identifier
MessageType	Message name[1]
MessageFormat	Physical format name[2]

Notes:

1. For details of the syntax of Message type, see "Specifying namespaces in the Message property" on page 2160.
2. The name that you specify for the physical format must match the name that you have defined for it. The default physical format names are Binary1, XML1, and Text1.

This ESQL procedure sets message properties to values passed in by the calling statement. You might find that you have to perform this task frequently, and you can use a procedure such as this in many different nodes and message flows. If you prefer, you can code ESQL that sets specific values.

```
CREATE PROCEDURE setMessageProperties(IN OutputRoot REFERENCE, IN setName char,
                 IN typeName char, IN formatName char) BEGIN
 /*********************************************************************
 * A procedure that sets the message properties
 *********************************************************************/
 set OutputRoot.Properties.MessageSet    = setName;
 set OutputRoot.Properties.MessageType   = typeName;
 set OutputRoot.Properties.MessageFormat = formatName;
END;
```

To set the output message domain, you can code ESQL statements that refer to the required domain in the second qualifier of the SET statement, the parser field. For example, the ESQL statement sets the domain to MRM:

```
SET OutputRoot.MRM.Field1 = 'field1 data';
```

This ESQL statement sets the domain to XMLNS:

```
SET OutputRoot.XMLNS.Field1 = 'field1 data';
```

Do not specify more than one domain in the ESQL for any single message. However, if you use PROPAGATE statements to generate several output messages, you can set a different domain for each message.

For information about the full list of elements in the Properties tree, see Data types for elements in the Properties subtree.

Differences exist in the way the Properties folder and the MQMD folder are treated with respect to which folder takes precedence for the same fields. For more information, see "Properties versus MQMD folder behavior for various transports" on page 506.

Accessing the local environment tree
The local environment tree has its own correlation name, `LocalEnvironment`, and you must use this name in all ESQL statements that refer to or set the content of this tree.

About this task

You can refer to and modify the information in the local environment tree. You can also extend the tree to contain information that you create yourself. You can create subtrees of this tree that you can use as a scratchpad or working area.

The message flow sets up information in two subtrees, `Destination` and `WrittenDestination`, below the `LocalEnvironment` root. You can refer to the content of both of these subtrees, and you can write to the `Destination` tree to influence the way in which the message flow processes your message. However, if you write to the `Destination` tree, follow the defined structure to ensure that the tree remains valid.

The `WrittenDestination` subtree contains the addresses to which the message has been written. Its name is fixed and it is created by the message flow when a message is propagated through the Out terminal of a request, output, or reply node. The subtree includes transport-specific information (for example, if the output message has been put to an IBM MQ queue, it includes the queue manager and queue names). You can use the following method to obtain information about the details of a message after it has been sent by the nodes:

- Connect a Compute node to the Out terminal.

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

If you want the local environment tree to be included in the output message that is propagated by the Compute node, you must set the Compute node property Compute mode to a value that includes the local environment (for example, All). If you do not, the local environment tree is not copied to the output message.

The information that you insert into DestinationData or Defaults depends on the characteristic of the corresponding node property:

- If a node property is represented by a check box (for example, New Message ID), set the Defaults or DestinationData element to Yes (equivalent to selecting the check box) or No (equivalent to clearing the check box).
- If a node property is represented by a drop-down list (for example, Transaction Mode), set the Defaults or DestinationData element to the appropriate character string (for example Automatic).
- If a node property is represented by a text entry field (for example, Queue Manager Name), set the Defaults or DestinationData element to the character string that you would enter in this field.

If necessary, configure the sending node to indicate where the destination information is. For example, for the output node MQOutput, set Destination Mode:

- If you set Destination Mode to Queue Name, the output message is sent to the queue identified in the output node properties Queue Name and Queue Manager Name. Destination is not referenced by the node.
- If you set Destination Mode to Destination List, the node extracts the destination information from the Destination subtree. If you use this value you can send a single message to multiple destinations, if you configure Destination and a single output node correctly. The node checks the node properties only if a value is not available in Destination (as described above).
- If you set Destination Mode to Reply To Queue, the message is sent to the reply-to queue identified in the MQMD in this message (field ReplyToQ). Destination is not referenced by the node.

To find more information about ESQL procedures that perform typical updates to the local environment, see "Populating Destination in the local environment tree" on page 1656. Review the ESQL statements in these procedures to see how to modify the local environment. You can use these procedures unchanged, or modify them for your own requirements.

To find more information about how to extend the contents of this tree for your own purposes, see "Using scratchpad areas in the local environment" on page 1655.

Using scratchpad areas in the local environment
The local environment tree includes a subtree called variables. This subtree is always created, but is never populated by the message flow. Use this area for your own purposes; for example, to pass information from one node to another. You can create other subtrees of the local environment tree.

About this task

The advantage of creating your own data in a scratchpad in the local environment is that this data can be propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, you can also include all or part of the local environment tree from the input message in the new output message.

To ensure that the information in the local environment is propagated further down the flow, the Compute mode property of the Compute node must be set to include the local environment as part of the output tree (for example, specify LocalEnvironment and Message). For further details about the Compute mode property, see Setting the mode.

However, any data updates or additions that you make in one node are not retained if the message moves backwards through the message flow (for example, if an exception is thrown). If you create your own data, and want that data to be preserved throughout the message flow, you must use the environment tree.

You can set values in the variables subtree in a Compute node and those values can be used later by another node (Compute, Database, or Filter) for some purpose that you determine when you configure the message flow.

The local environment is not in scope in a Compute node, therefore you must use `InputLocalEnvironment` and `OutputLocalEnvironment` instead. For example, you might use the scratchpad in the local environment to propagate the destination of an output message to subsequent nodes in a message flow. Your first Compute node determines that the output messages from this message flow must go to IBM MQ queues. Include the following ESQL to insert this information into the local environment by setting the value of *OutputLocation* in the OutputLocalEnvironment:

```
SET OutputLocalEnvironment.Variables.OutputLocation = 'MQ';
```

Your second Compute node can access this information from its input message. In the ESQL in this node, use the correlation name InputLocalEnvironment to identify the local environment tree in the input message that contains this data. The following ESQL sets *queueManagerName* and *queueName* based on the content of *OutputLocation* in the local environment, by using InputLocalEnvironment:

```
IF InputLocalEnvironment.Variables.OutputLocation = 'MQ' THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueManagerName = 'myQManagerName';
   SET OutputLocalEnvironment.Destination.MQ.DestinationData.queueName = 'myQueueName';
END IF;
```

In the example, *queueManagerName* and *queueName* are set for the Destination subtree in the output message. You must set the `Compute mode` of the second Compute node to include the local environment tree in the output message. Configure the MQOutput node to use the destination list that you have created in the local environment tree by setting the `Destination Mode` property to `Destination List`.

For information about the full list of elements in the DestinationData subtree, see Data types for elements in the MQ DestinationData subtree.

Populating Destination in the local environment tree
Use the Destination subtree to set up the target destinations that are used by output nodes, the HTTPRequest node, the SOAPRequest node, the SOAPAsyncRequest node, and the RouteToLabel node. The following examples show how you can create and use an ESQL procedure to perform the task of setting up values for each of these uses.

About this task

Copy and use these procedures as shown, or you can modify or extend them to perform similar tasks.

If you are creating this ESQL code for a Compute node, you must configure the node by setting the `Compute Mode` property so that it has access to the local environment tree in the output message. You must select one of the three values `LocalEnvironment`, `LocalEnvironment And Message`, or `All`.

Adding a queue name for the MQOutput node with the `Destination Mode` property set to `Destination List`

```
CREATE PROCEDURE addToMQDestinationList(IN LocalEnvironment REFERENCE, IN newQueue char) BEGIN
  /******************************************************************************
  * A procedure that adds a queue name to the MQ destination list in the local environment.
  * This list is used by an MQOutput node that has its mode set to Destination list.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the queue to be added to the list
  *
  ******************************************************************************/
DECLARE I INTEGER CARDINALITY(LocalEnvironment.Destination.MQ.DestinationData[]);
  IF I = 0 THEN
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName = newQueue;
  ELSE
   SET OutputLocalEnvironment.Destination.MQ.DestinationData[I+1].queueName = newQueue;
  END IF;
 END;
```

For full details of these elements, see Data types for elements in the MQ DestinationData subtree.

Changing the default URL for a SOAPRequest node or a SOAPAsyncRequest node request

```
CREATE PROCEDURE overrideDefaultSOAPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /**************************************************************************
  * A procedure that changes the URL to which the SOAPRequest node or
  * SOAPAsyncRequest node sends the request.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue:  the URL to which to send the request.
  *
  **************************************************************************/
SET OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.WebServiceURL = newUrl;
END;
```

Changing the default URL for an HTTPRequest node request

```
CREATE PROCEDURE overrideDefaultHTTPRequestURL(IN LocalEnvironment REFERENCE, IN newUrl char) BEGIN
  /**************************************************************************
  * A procedure that changes the URL to which the HTTPRequest node sends the request.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN queue: the URL to which to send the request.
  *
  **************************************************************************/
  SET OutputLocalEnvironment.Destination.HTTP.RequestURL = newUrl;
END;
```

Adding a label for the RouteToLabel node

```
CREATE PROCEDURE addToRouteToLabelList(IN LocalEnvironment REFERENCE, IN newLabel char) BEGIN
  /**************************************************************************
  * A procedure that adds a label name to the RouteToLabel list in the local environment.
  * This list is used by a RoteToLabel node.
  *
  * IN LocalEnvironment: the LocalEnvironment to be modified.
  * IN label:  the label to be added to the list
  *
  **************************************************************************/
 IF LocalEnvironment.Destination.RouterList.DestinationData is null THEN
     SET OutputLocalEnvironment.Destination.RouterList.DestinationData."label" = newLabel;
  ELSE
     CREATE LASTCHILD OF LocalEnvironment.Destination.RouterList.DestinationData
     NAME 'label' VALUE newLabel;
  END IF;
END;
```

Setting up JMS destination lists

You can configure a JMSOutput node to send to multiple JMS Queues, or to publish to multiple JMS Topics by using a destination list that is created in the local environment tree by a transformation node. The following example shows how to set up JMS destination lists in the local environment tree:

```
CREATE PROCEDURE CreateJMSDestinationList() BEGIN
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[1]  = 'jndi://
TestDestQueue1';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[2]  = 'jndi://
TestDestQueue2';
  SET OutputLocalEnvironment.Destination.JMSDestinationList.DestinationData[3]  = 'jndi://
TestDestQueue3';
END;
```

Accessing the environment tree

The environment tree has its own correlation name, Environment, and you must use this name in all ESQL statements that refer to, or set, the content of this tree.

About this task

The environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it, nor uses its contents. You can use this tree for your own purposes, for example, to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The advantage of creating your own data in environment is that this data is propagated as part of the logical tree to subsequent nodes in the message flow. If you create a new output message in a Compute node, the environment tree is also copied from the input message to the new output message. (In contrast to the local environment tree, which is only included in the output message if you explicitly request that it is).

Only one environment tree is present for the duration of the message flow. Any data updates, or additions, that you make in one node are retained, and all of the nodes in the message flow have access to the latest copy of this tree. Even if the message flows back through the message flow (for example, if an exception is thrown, or if the message is processed through the second terminal of the FlowOrder node), the latest state is retained. (In contrast to the local environment tree, which reverts to its previous state if the message flows back through the message flow.)

You can use this tree for any purpose you choose. For example, you can use the following ESQL statements to create fields in the tree:

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

This information is now available to all nodes to which a message is propagated, regardless of their relative position in the message flow.

Accessing the ExceptionList tree using ESQL
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

About this task

This tree is created with the logical tree when an input message is parsed. It is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in Compute, Database, and Filter nodes, and you can update it in a Compute node. You must use the appropriate correlation name; ExceptionList for a Database or Filter node, and InputExceptionList for a Compute node.

You might want to access this tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception, for example one that you have explicitly generated using an ESQL THROW statement, or one that the integration node has generated.

The following ESQL shows how you can access the ExceptionList and process each child that it contains:

```
-- Declare a reference for the ExceptionList
-- (in a Compute node use InputExceptionList)
DECLARE start REFERENCE TO ExceptionList.*[1];

-- Loop through the exception list children
WHILE start.Number IS NOT NULL DO
   -- more ESQL

   -- Move start to the last child of the field to which it currently points
   MOVE start LASTCHILD;
END WHILE;
```

The following example shows an extract of ESQL that has been coded for a Compute node to loop through the exception list to the last (nested) exception description and extract the error number. This error relates to the original cause of the problem and normally provides the most precise information. Subsequent action taken by the message flow can be decided by the error number retrieved in this way.

```
CREATE PROCEDURE getLastExceptionDetail(IN InputTree reference,OUT messageNumber integer,
OUT messageText char)
    /*****************************************************************************
  * A procedure that will get the details of the last exception from a message
  * IN InputTree:  The incoming exception list
  * IN messageNumber:  The last message numberr.
  * IN messageText: The last message text.
  *****************************************************************************/
   BEGIN
        -- Create a reference to the first child of the exception list
        declare ptrException reference to InputTree.*[1];
        -- keep looping while the moves to the child of exception list work
  WHILE lastmove(ptrException) DO
   -- store the current values for the error number and text
   IF ptrException.Number is not null THEN
          SET messageNumber = ptrException.Number;
          SET messageText = ptrException.Text;
     END IF;
     -- now move to the last child which should be the next exceptionlist
   move ptrException lastchild;
  END WHILE;
 END;
```

The following ESQL example shows how you can convert the exception list to character format:

```
DECLARE tmp ROW;
CREATE LASTCHILD OF tmp DOMAIN('XMLNSC') NAME 'XMLNSC';
SET tmp.XMLNSC.ExceptionList = InputExceptionList;
SET tmp.BLOB = ASBITSTREAM(tmp.XMLNSC.ExceptionList OPTIONS
FolderBitStream CCSID 1208);
DECLARE FULLEXCEPTION CHAR CAST(tmp.BLOB AS CHAR CCSID 1208);
LOG USER TRACE VALUES('FULLEXCEPTION got is ',FULLEXCEPTION);
```

For information on accessing the ExceptionList tree using Java, see “Accessing the ExceptionList tree using Java” on page 1780.

Transforming from one data type to another
Code ESQL functions and statements to transform messages and data types in many ways.

About this task

The following topics provide guidance:

- “Casting data from message fields” on page 1659
- “Converting code page and message encoding” on page 1660
- “Converting EBCDIC NL to ASCII CR LF” on page 1662
- “Changing message format” on page 1665

Casting data from message fields
You can use the CAST function to transform the data type of one value to match the data type of the other. For example, you can use the CAST function when you process generic XML messages. All fields in an XML message have character values, so if you want to perform arithmetic calculations or datetime comparisons, for example, you must convert the string value of the field into a value of the appropriate type using CAST.

About this task

When you compare an element with another element, variable or constant, ensure that the value with which you are comparing the element is consistent (for example, character with character). If the values are not consistent, the integration node generates a runtime error if it cannot provide an implicit casting to resolve the inconsistency. For details of what implicit casts are supported, see Implicit casts.

In the Example message, the field InvoiceDate contains the date of the invoice. If you want to refer to or manipulate this field, you must CAST it to the correct format first. For example, to refer to this field in a test:

```
IF CAST(Body.Invoice.InvoiceDate AS DATE) = CURRENT_DATE THEN
```

This converts the string value of the InvoiceDate field into a date value, and compares it to the current date.

Another example is casting from integer to character:

```
DECLARE I INTEGER 1;
DECLARE C CHARACTER;

-- The following statement generates an error
SET C = I;

-- The following statement is valid
SET C = CAST(I AS CHARACTER);
```

Converting code page and message encoding
You can use ESQL within a Compute node to convert data for code page and message encoding.

About this task

If your message flow is processing IBM MQ messages, you can use IBM MQ facilities (including get and put options and IBM MQ data conversion exits) to provide these conversions. If you are not processing IBM MQ messages, or you choose not to use IBM MQ facilities, you can use IBM App Connect Enterprise facilities by coding the appropriate ESQL in a Compute node in your message flow.

The contents of the MQMD, the MQRFH2, and the message body of a message in the MRM domain that has been modeled with a CWF physical format can be subject to code page and encoding conversion. The contents of a message body of a message in the XML, XMLNS, and JMS domains, and those messages in the MRM domain that have been modeled with an XML or TDS physical format, are treated as strings. Only code page conversion applies; no encoding conversion is required.

For messages in the MRM domain modeled with a CWF physical format, you can set the MQMD CCSID and Encoding fields of the output message, plus the CCSID and Encoding of any additional headers, to the required target value.

For messages in the MRM domain modeled with an XML or TDS physical format, you can set the MQMD CCSID field of the output message, plus the CCSID of any additional headers. XML and TDS data is handled as strings and is therefore subject to CCSID conversion only.

An example IBM MQ message has an MQMD header, an MQRFH2 header, and a message body. To convert this message to a mainframe CodedCharSetId and Encoding, code the following ESQL in the Compute node:

```
SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQRFH2.CodedCharSetId = 500;
SET OutputRoot.MQRFH2.Encoding = 785;
```

The following example illustrates what you must do to modify a CWF message so that it can be passed from IBM App Connect Enterprise to IMS on z/OS.

Procedure

1. You have defined the input message in XML and are using an MQRFH2 header. Remove the header before passing the message to IMS.
2. The message passed to IMS must have MQIIH header, and must be in the z/OS code page.

   This message is modeled in the MRM and has the name IMS1. Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.

3. The message received from IMS is also defined in the MRM and has the name IMS2.

   Define the PIC X fields in this message as logical type string for conversions between EBCDIC and ASCII to take place. If the fields are binary logical type, no data conversion occurs; binary data is ignored when a CWF message is parsed by the MRM parser.
4. Convert the reply message to the Windows code page.

   The MQIIH header is retained on this message.
5. You have created a message flow that contains the following nodes:

   :

   a. The outbound flow, **MQInput1 --> Compute1 --> MQOutput1**.

   b. The inbound flow, **MQInput2 --> Compute2 --> MQOutput2**.
6. Code ESQL in Compute1 (outbound) node as follows, specifying the relevant MessageSet ID.

   This code shows the use of the default CWF physical layer name. You must use the name that matches your model definitions. If you specify an incorrect value, the integration node fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J - 1 DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 500;
SET OutputRoot.MQMD.Encoding = 785;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.Version = 1;
SET OutputRoot.MQIIH.StrucLength = 84;
SET OutputRoot.MQIIH.Encoding = 785;
SET OutputRoot.MQIIH.CodedCharSetId = 500;
SET OutputRoot.MQIIH.Format = 'MQIMSVS ';
SET OutputRoot.MQIIH.Flags = 0;
SET OutputRoot.MQIIH.LTermOverride = '        ';
SET OutputRoot.MQIIH.MFSMapName = '        ';
SET OutputRoot.MQIIH.ReplyToFormat = 'MQIMSVS ';
SET OutputRoot.MQIIH.Authenticator = '        ';
SET OutputRoot.MQIIH.TranInstanceId = X'00000000000000000000000000000000';
SET OutputRoot.MQIIH.TranState = ' ';
SET OutputRoot.MQIIH.CommitMode = '0';
SET OutputRoot.MQIIH.SecurityScope = 'C';
SET OutputRoot.MQIIH.Reserved = ' ';
SET OutputRoot.MRM.e_elen08 = 30;
SET OutputRoot.MRM.e_elen09 = 0;
SET OutputRoot.MRM.e_string08 = InputBody.e_string01;
SET OutputRoot.MRM.e_binary02 = X'31323334353637383940';
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS1';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

   The use of a variable, J, that is initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

7. Create ESQL in Compute2 (inbound) node as follows, specifying the relevant MessageSet ID.

   This code shows the use of the default CWF physical layer name. You must use the name that matches your model definition. If you specify an incorrect value, the integration node fails with message BIP5431.

```
-- Loop to copy message headers
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

SET OutputRoot.MQMD.CodedCharSetId = 437;
SET OutputRoot.MQMD.Encoding = 546;
SET OutputRoot.MQMD.Format = 'MQIMS   ';
SET OutputRoot.MQIIH.CodedCharSetId = 437;
SET OutputRoot.MQIIH.Encoding = 546;
SET OutputRoot.MQIIH.Format = '        ';
SET OutputRoot.MRM = InputBody;
SET OutputRoot.Properties.MessageSet = 'DHCJOEG072001';
SET OutputRoot.Properties.MessageType = 'IMS2';
SET OutputRoot.Properties.MessageFormat = 'Binary1';
```

Results

You do not have to set any specific values for the MQInput1 node properties, because the message and message set are identified in the MQRFH2 header, and no conversion is required.

You must set values for message domain, set, type, and format in the MQInput node for the inbound message flow (MQInput2). You do not need to set conversion parameters.

One specific situation in which you might need to convert data in one code page to another is when messages contain newline characters and are passed between EBCDIC and ASCII systems. The required conversion for this situation is described in “Converting EBCDIC NL to ASCII CR LF” on page 1662.

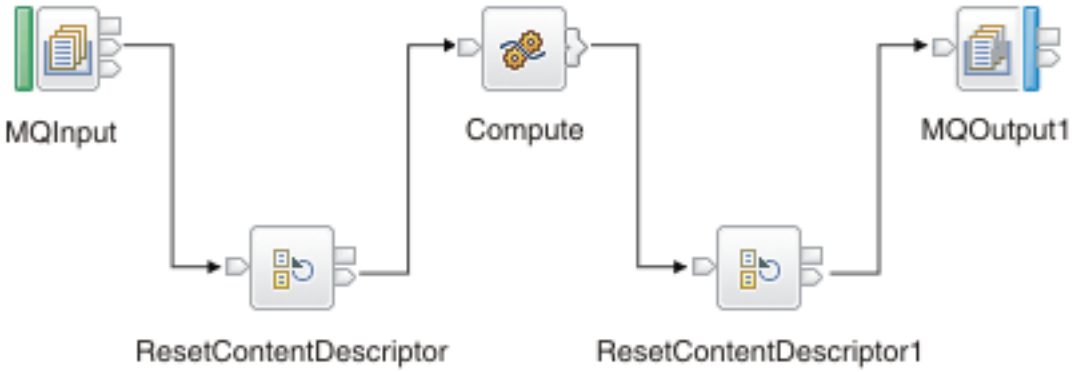
Converting EBCDIC NL to ASCII CR LF

You might want to change newline (NL) characters in a text message to carriage return (CR) and line feed (LF) character pairs. This example shows one way in which you can convert these characters.

About this task

This conversion might be useful if messages from an EBCDIC platform (for example, using CCSID 1047) are sent to an ASCII platform (for example, using CCSID 437). Problems can arise because the EBCDIC NL character hex '15' is converted to the undefined ASCII character hex '7F'. The ASCII code page has no corresponding code point for the NL character.

In this example, a message flow is created that interprets the input message as a message in the BLOB domain. This message is passed into a ResetContentDescriptor node to reset the data to a message in the MRM domain. The message is called msg_nl (a set of repeating string elements delimited by EBCDIC NL characters). A Compute node is then used to create an output based on another message in the MRM domain called msg_crlf (a set of repeating string elements delimited by CR LF pairs). The message domain is then changed back to BLOB in another ResetContentDescriptor node. This message flow is shown in the following diagram.



The following instructions show how to create the messages and configure the message flow.

Procedure

1. Create the message models for the messages in the MRM domain:
   a. Create a message set project called myProj.
   b. Create a message set called myMessageSet with a TDS physical format (the default name is Text1).
   c. Create an element string1 of type xsd:string.
   d. Create a complex type called t_msg_nl and specify the following complex type properties:
      - `Composition` = Ordered Set
      - `Content Validation` = Closed
      - `Data Element Separation` = All Elements Delimited
      - `Delimiter` = <U+0085> (hex '0085' is the UTF-16 representation of an NL character)
      - `Repeat` = Yes
      - `Min Occurs` = 1
      - `Max Occurs` = 50 (the text of the message is assumed to consist of no more than 50 lines)
   e. Add Element string1, and set the following property:
      - `Repeating Element Delimiter` = <U+0085>
   f. Create a Message msg_nl, and set its associated complex type to t_msg_nl
   g. Create a complex type called t_msg_crlf, and specify the following complex type properties:
      - `Composition` = Ordered Set
      - `Content Validation` = Closed
      - `Data Element Separation` = All Elements Delimited
      - `Delimiter` = <CR><LF> (<CR> and <LF> are the mnemonics for the CR and LF characters)
      - `Repeat` = Yes
      - `Min Occurs` = 1
      - `Max Occurs` = 50
   h. Add Element string1, and set the following property:
      - `Repeating Element Delimiter` = <CR><LF>
   i. Create a Message msg_crlf, and set complex type to t_msg_crlf.

2. Configure the message flow shown in the previous figure:
   a. Start with the MQInput node:
      - Set `Message Domain` = BLOB
      - Set `Queue Name` = <Your input message queue name>
   b. Add the ResetContentDescriptor node, connected to the Out terminal of the MQInput node:
      - Set `Message Domain` = MRM
      - Select `Reset Message Domain`
      - Set `Message Set` = <Your Message Set ID> (this field has a maximum of 13 characters)
      - Select `Reset Message Set`
      - Set `Message Type` = msg_nl
      - Select `Reset Message Type`
      - Set `Message Format` = Text1
      - Select `Reset Message Format`
   c. Add the Compute node, connected to the Out terminal of the ResetContentDescriptor node:
      - Enter a name for the ESQL `Module` for this node, or accept the default (<message flow name>_Compute).
      - Right-click the Compute node, and select **Open ESQL**. Add the following ESQL code in the module:

```
-- Declare local working variables
DECLARE I INTEGER 1;
DECLARE J INTEGER CARDINALITY(InputRoot.*[]);

-- Loop to copy all message headers from input to output message
WHILE I < J DO
  SET OutputRoot.*[I] = InputRoot.*[I];
  SET I=I+1;
END WHILE;

-- Set new output message type which uses CRLF delimiter
SET OutputRoot.Properties.MessageType = 't_msg_crlf';

-- Loop to copy each instance of string1 child within message body
SET I = 1;
SET J = CARDINALITY("InputBody"."string1"[]);
WHILE I <= J DO
  SET "OutputRoot"."MRM"."string1"[I] = "InputBody"."string1"[I];
  SET I=I+1;
END WHILE;
```

      The use of a variable, J, initialized to the value of the cardinality of the existing headers in the message, is more efficient than calculating the cardinality on each iteration of the loop, which happens if you code the following WHILE statement:

```
WHILE I < CARDINALITY(InputRoot.*[]) DO
```

   d. Add the ResetContentDescriptor1 node, connected to the Out terminal of the Compute node:
      - Set `Message Domain` = BLOB
      - Select `Reset Message Domain`.
   e. Finally, add the MQOutput node, connected to the Out terminal of the ResetContentDescriptor1 node. Configure its properties to direct the output message to the required queue or queues.

Changing message format
Use the Compute node to copy part of an input message to an output message. The results of such a copy depend on the type of input and output parsers involved.

Before you begin

If you are copying between unlike parsers, some attributes of the original message might be lost. To ensure that you copy every attribute, copy the message to the same parser.

Like parsers

About this task

Where both the source and target messages have the same folder structure at root level, a *like-parser-copy* is performed. For example:

```
SET OutputRoot.MQMD = InputRoot.MQMD;
```

This statement copies all the children in the MQMD folder of the input message to the MQMD folder of the output message.

Another example of a tree structure that supports a like-parser-copy is:

```
SET OutputRoot.XMLNS.Data.Account = InputRoot.XMLNS.Customer.Bank.Data;
```

To transform an input message in the MRM domain to an output message also in the MRM domain, you can use either the Compute or the Mapping node. The Mapping node can interpret the action that is required because it knows the format of both messages. Content Assist in the ESQL module for the Compute node can also use the message definitions for those messages. If the messages are not in the same namespace, you must use the Compute node.

To use Content Assist with message references, you must set up a project reference from the project containing the ESQL to the project containing the message set. For information about setting up a project reference, see “Referencing resources in other libraries” on page 1976.

If both input and output messages are not in the MRM domain, you must use the Compute node and specify the structure of the messages yourself.

Unlike parsers

About this task

Where the source and target messages have different folder structures at root level, you cannot make an exact copy of the message source. Instead, the *unlike-parser-copy* views the source message as a set of nested folders terminated by a leaf name-value pair. For example, copying the following message from XML to MRM:

```
<Name3><Name31>Value31</Name31>Value32</Name3>
```

produces a name element Name3, and a name-value element called Name31 with the value Value31. The second XML pcdata (Value32) cannot be represented and is discarded.

The unlike-parser-copy scans the source tree, and copies folders, also known as name elements, and leaf name-value pairs. Everything else, including elements flagged as *special* by the source parser, is not copied.

An example of a tree structure that results in an unlike-parser-copy is:

```
SET OutputRoot.DFDL.Data.Account = InputRoot.XMLNSC.Data.Account;
```

If the algorithm used to make an unlike-parser-copy does not suit your tree structure, youmust further qualify the source field to restrict the amount of the tree that is copied.

Be careful when you copy information from input messages to output messages in different domains. You might code ESQL that creates a message structure or content that is not consistent with the rules of the parser that processes the output message. This action can result in an output message not being created, or being created with unexpected content. If you believe that the output message generated by a particular message flow does not contain the correct content, or have the expected form, check the ESQL that creates the output message, and look for potential mismatches of structure, field types, field names, and field values.

When copying trees between unlike parsers, it might be necessary to set the MessageSet, MessageType, and MessageFormat fields in the output Properties folder. Which fields need setting is dependent on the target parser. For example, if the target parser is MRM and the message set has been defined with a CWF format, the following commands are required to copy a message body created by the XMLNSC parser to a body owned by the MRM parser:

```
-- Copy message to the output, moving from XMLNSC to MRM domains
SET OutputRoot.MRM = InputRoot.XMLNSC.rootElement;

-- Set the CWF format for output by the MRM domain
SET OutputRoot.Properties.MessageType = '<MessageTypeName>';
SET OutputRoot.Properties.MessageSet = '<MessageSetName>';
SET OutputRoot.Properties.MessageFormat = 'CWF';
```

Interaction with databases using ESQL
Use ESQL statements and functions to read from, write to, and modify databases from your message flows.

About this task

ESQL has a number of statements and functions for accessing databases:

- The CALL statement invokes a stored procedure.
- The DELETE FROM statement removes rows from a database table.
- The INSERT statement adds a row to a database table.
- The PASSTHRU function can be used to make complex selections.
- The PASSTHRU statement can be used to invoke administrative operations (for example, creating a table).
- The SELECT function retrieves data from a table.
- The UPDATE statement changes one or more values stored in zero or more rows.

You can access databases from Compute, Database, DatabaseInput, and Filter nodes. The same ESQL functions and procedures are supported in all these nodes.

You can use the data in the databases to update or create messages, or use the data in the message to update or create data in the databases.

Select **Throw exception on database error** and **Treat warnings as errors**, and set **`Transaction`** to `Automatic` on each node that access a database, to provide maximum flexibility.

For information about configuring the integration node and the database to support access from message flows, see “Accessing databases from ESQL” on page 1135.

Referencing columns in a database

About this task

While the standard SQL SELECT syntax is supported for queries to an external database, there are a number of points to be borne in mind. You must prefix the name of the table with the keyword Database to indicate that the SELECT is to be targeted at the external database, rather than at a repeating structure in the message.

The basic form of database SELECT is:

```
SELECT ...
  FROM Database.TABLE1
  WHERE ...
```

If necessary, you can specify a schema name:

```
SELECT ...
  FROM Database.SCHEMA.TABLE1
  WHERE ...
```

where SCHEMA is the name of the schema in which the table TABLE1 is defined. Include the schema if the user ID under which you are running does not match the schema. For example, if your userID is USER1, the expression Database.TABLE1 is equivalent to Database.USER1.TABLE1. However, if the schema associated with the table in the database is db2admin, you must specify Database.db2admin.TABLE1. If you do not include the schema, and this does not match your current user ID, the integration node generates a runtime error when a message is processed by the message flow.

If, as in the two previous examples, a data source is not specified, TABLE1 must be a table in the default database specified by the node's `data source` property. To access data in a database other than the default specified on the node's `data source` property, you must specify the data source explicitly. For example:

```
SELECT ...
  FROM Database.DataSource.SCHEMA.TABLE1
  WHERE ...
```

Qualify references to column names with either the table name or the correlation name defined for the table by the FROM clause. So, where you could normally execute a query such as:

```
SELECT column1, column2 FROM table1
```

you must write one of the following two forms:

```
SELECT T.column1, T.column2 FROM Database.table1 AS T

SELECT table1.column1, table1.column2 FROM Database.table1
```

This is necessary in order to distinguish references to database columns from any references to fields in a message that might also appear in the SELECT:

```
SELECT T.column1, T.column2 FROM Database.table1
  AS T WHERE T.column3 = Body.Field2
```

You can use the AS clause to rename the columns returned. For example:

```
SELECT T.column1 AS price, T.column2 AS item
  FROM Database.table1 AS T WHERE...
```

The standard select all SQL option is supported in the SELECT clause. If you use this option, you must qualify the column names with either the table name or the correlation name defined for the table. For example:

```
SELECT T.* FROM Database.Table1 AS T
```

When you use ESQL procedure and function names within a database query, the positioning of these within the call affects how these names are processed. If it is determined that the procedure or function affects the results returned by the query, it is not processed as ESQL and is passed as part of the database call.

This applies when attempting to use a function or procedure name with the column identifiers within the SELECT statement.

For example, if you use a CAST statement on a column identifier specified in the Select clause, this is used during the database query to determine the data type of the data being returned for that column.

An ESQL CAST is not performed to that ESQL data type, and the data returned is affected by the database interaction's interpretation of that data type.

If you use a function or procedure on a column identifier specified in the WHERE clause, this is passed directly to the database manager for processing.

The examples in the subsequent topics illustrate how the results sets of external database queries are represented in IBM App Connect Enterprise. The results of database queries are assigned to fields in a message using a Compute node.

A column function is a function that takes the values of a single column in all the selected rows of a table or message and returns a single scalar result.

Selecting data from database columns
You can configure a Compute, Filter, or Database node to select data from database columns and include it in an output message.

About this task

The following example assumes that you have a database table called USERTABLE with two char(6) data type columns (or equivalent), called Column1 and Column2. The table contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

Configure the Compute, Filter, or Database node to identify the database in which you have defined the table. For example, if you are using the default database (specified on the `data source` property of the node), right-click the node, select **Open ESQL**, and code the following ESQL statements in the module for this node:

```
SET OutputRoot = InputRoot;
DELETE FIELD OutputRoot.*[<];
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1, T.Column2 FROM Database.USERTABLE AS T);
```

This ESQL produces the following output message:

```
<Test>
   <Result>
      <Column1>value1</Column1>
      <Column2>value2</Column2>
   </Result>
   <Result>
      <Column1>value3</Column1>
      <Column2>value4</Column2>
   </Result>
</Test>
```

Figure 4. Output message

To trigger the SELECT, send a trigger message with an XML body that is of the following form:

```
<Test>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
   <Result>
      <Column1></Column1>
      <Column2></Column2>
   </Result>
</Test>
```

The exact structure of the XML is not important, but the enclosing tag must be <Test> to match the reference in the ESQL. If the enclosing tag is not <Test>, the ESQL statements result in top-level enclosing tags being formed, which is not valid XML.

If you want to create an output message that includes all the columns of all the rows that meet a particular condition, use the SELECT statement with a WHERE clause:

```
-- Declare and initialize a variable to hold the
--      test vaue (in this case the surname Smith)
DECLARE CurrentCustomer STRING 'Smith';

-- Loop through table records to extract matching information
SET OutputRoot.XML.Invoice[] =
    (SELECT R FROM Database.USERTABLE AS R
               WHERE R.Customer.LastName = CurrentCustomer
    );
```

The message fields are created in the same order as the columns occur in the table.

If you are familiar with SQL in a database environment, you might expect to code SELECT *. This syntax is not accepted by the integration node, because you must start all references to columns with a correlation name to avoid ambiguities with declared variables. Also, if you code SELECT I.*, this syntax is accepted by the integration node, but the * is interpreted as the first child element, not all elements, as you might expect from other database SQL.

The assignment of the result set of a database into a parser-owned message tree requires the result set to exactly match the message definition. Because the generic XML parser is self-defining, the example creates a new subtree off the Invoice folder, and the parser can parse the new elements in the subtree. If the structure of the result set exactly matches the message definition, the result set can be assigned directly into the `OutputRoot` message body tree.

If the structure of the result set does not exactly match the MRM message definition, you must first assign the result set into a ROW data type, or an Environment tree that does not have a parser associated with it.

The required data can then be assigned to `OutputRoot` to build a message tree that conforms to the message definition.

Selecting data from a table in a case-sensitive database system

About this task

If the database system is case sensitive, you must use an alternative approach. This approach is also necessary if you want to change the name of the generated field to something different:

```
SET OutputRoot = InputRoot;
SET OutputRoot.XML.Test.Result[] =
  (SELECT T.Column1 AS Column1, T.Column2 AS Column2
  FROM Database.USERTABLE AS T);
```

This example produces the output message as shown in Figure 4 on page 1668. Ensure that references to the database columns (in this example, T.Column1 and T.Column2) are specified in the correct case to match the database definitions exactly. If you do not match the database definitions exactly (for example if you specify T.COLUMN1), the integration node generates a runtime error. `Column1` and `Column2` are used in the SELECT statement to match the columns that you have defined in the database, although you can use any values here; the values do not have to match.

Selecting bitstream data from a database

About this task

These samples show how to retrieve XML bitstream data from a database, and include it in an output message. See INSERT statement for examples that show how you can insert bitstream data into a database.

Example

In the following example, bitstream data is held in a database column with a BLOB data type. The database table used in the example (TABLE1) is the one created in the INSERT statement examples, and the table contains the following columns:

- MSGDATA
- MSGCCSID
- MSGENCODING

If the bit stream from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain without any alteration.

In the following example, the message data, along with the MQMD header, is held in a database column with a BLOB data type. To re-create the message tree, including the MQMD header, from the bit stream, you can use a CREATE statement with a PARSE clause and DOMAIN('MQMD'). The output message can then be modified by the message flow:

```
SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;

IF LASTMOVE(resultRef) THEN

  DECLARE outMsg BLOB resultRef.MSGDATA ;
  DECLARE outCCSID INT resultRef.MSGCCSID;
  DECLARE outEncoding INT resultRef.MSGENCODING;
  DECLARE outMsgPriority INT resultRef.MSGPRIORITY;
  DECLARE outMsgSeqNum INT resultRef.MSGSEQNUMBER;

  SET OutputRoot.Properties.CodedCharSetId = outCCSID;
  SET OutputRoot.Properties.Encoding = outEncoding ;

  CREATE LASTCHILD OF OutputRoot DOMAIN('MQMD') PARSE(outMsg, outEncoding, outCCSID);

    SET OutputRoot.MQMD.Version = MQMD_VERSION_2;


  SET OutputRoot.MQMD.Priority = outMsgPriority;
  SET OutputRoot.MQMD.MsgSeqNumber = outMsgSeqNum;

  DECLARE HDRL INT ;
  SET HDRL = LENGTH(BITSTREAM(OutputRoot.MQMD));
  CREATE FIELD OutputRoot."BLOB"."BLOB";
  DECLARE MSGB BLOB;
  SET MSGB = SUBSTRING(outMsg FROM HDRL +1);
  SET OutputRoot."BLOB"."BLOB" = MSGB;

END IF;
```

If you want to interrogate or manipulate a bit stream extracted from a database, you must re-create the original message tree. To re-create the XML message tree from the bit stream, you can use a CREATE statement with a PARSE clause. The output message can then be modified by the message flow.

For example, you might create a database table by using the following statement:

```
INSERT INTO Database.TABLE1(MSGDATA, MSGENCODING, MSGCCSID)
   VALUES (msgBitStream, inEncoding, inCCSID);
```

The following code snippet shows how to re-create the message tree in the XMLNS domain by using the data read from the table:

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB resultRef.MSGDATA;
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
    -- Now modify the message tree fields
```

```
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. If the bitstream data from the database does not need to be interrogated or manipulated by the message flow, the output message can be constructed in the BLOB domain, without any alteration.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = resultRef.MSGENCODING;
    SET OutputRoot.BLOB.BLOB = outMsg;
  END IF;
```

In the following example, the data is held in a database column with a character data type, such as CHAR or VARCHAR. A cast is used to convert the data extracted from the database into BLOB format. To manipulate or interrogate this data within the message flow, you must re-create the original message tree. In this example, a CREATE statement with a PARSE clause is used to re-create the XML message tree in the XMLNS domain.

```
CALL CopyMessageHeaders();
  SET Environment.Variables.DBResult = THE( SELECT T.* FROM Database.TABLE1 AS T);
  DECLARE resultRef REFERENCE TO Environment.Variables.DBResult;
  IF LASTMOVE(resultRef) THEN
    DECLARE outCCSID INT resultRef.MSGCCSID;
    DECLARE outEncoding INT resultRef.MSGENCODING;
    DECLARE outMsg BLOB CAST(resultRef.MSGDATA AS BLOB CCSID outCCSID);
    SET OutputRoot.Properties.CodedCharSetId = outCCSID;
    SET OutputRoot.Properties.Encoding = outEncoding;
    CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNS') PARSE(outMsg, outEncoding, outCCSID);
    -- Now modify the message tree fields
    SET OutputRoot.XMLNS.A.B = 4;
    SET OutputRoot.XMLNS.A.E = 5;
  END IF;
```

Accessing multiple database tables
You can refer to multiple tables that you have created in the same database. Use the FROM clause on the SELECT statement to join the data from the two tables.

About this task

The following example assumes that you have two database tables called USERTABLE1 and USERTABLE2. Both tables have two char(6) data type columns (or equivalent).

USERTABLE1 contains two rows:

	Column1	**Column2**
Row 1	value1	value2
Row 2	value3	value4

USERTABLE2 contains two rows:

	Column3	**Column4**
Row 1	value5	value6
Row 2	value7	value8

All tables referenced by a single SELECT function must be in the same database. The database can be either the default (specified on the Data Source property of the node) or another database (specified on the FROM clause of the SELECT function).

Configure the Compute, Database, or Filter node that you are using to identify the database in which you have defined the tables. For example, if you are using the default database, right-click the node, select **Open ESQL**, and code the following ESQL statements in the module for this node:

```
SET OutputRoot.XML.Test.Result[] =
         (SELECT A.Column1 AS FirstColumn,
                 A.Column2 AS SecondColumn,
                 B.Column3 AS ThirdColumn,
                 B.Column4 AS FourthColumn
            FROM Database.USERTABLE1 AS A,
                 Database.USERTABLE2 AS B
           WHERE A.Column1 = 'value1' AND
                 B.Column4 = 'value8'
         );
```

This code results in the following output message content:

```
<Test>
  <Result>
    <FirstColumn>value1</FirstColumn>
    <SecondColumn>value2</SecondColumn>
    <ThirdColumn>value7</ThirdColumn>
    <FourthColumn>value8</FourthColumn>
  </Result>
</Test>
```

This example shows how to access data from two database tables. You can code more complex FROM clauses to access multiple database tables (although all the tables must be in the same database). You can also refer to one or more message trees, and can use SELECT to join tables with tables, messages with messages, or tables with messages. “Joining data from messages and database tables” on page 1691 provides an example of how to merge message data with data in a database table.

If you specify an ESQL function or procedure on the column identifier in the WHERE clause, it is processed as part of the database query and not as ESQL.

Consider the following example:

```
  SET OutputRoot.XML.Test.Result =
     THE(SELECT ITEM T.Column1 FROM Database.USERTABLE1 AS T
     WHERE UPPER(T.Column2) = 'VALUE2');
```

This code attempts to return the rows where the value of `Column2` converted to uppercase is `VALUE2`. However, only the database manager can determine the value of `T.Column2` for any given row, therefore it cannot be processed by ESQL before the database query is issued, because the WHERE clause determines the rows that are returned to the message flow.

Therefore, the UPPER is passed to the database manager to be included as part of its processing. However, if the database manager cannot process the token within the SELECT statement, an error is returned.

Changing database content
You can use Compute, Database, and Filter nodes to change the contents of a database by updating, inserting, or deleting data.

About this task

The following ESQL code includes statements that show all three operations. This code is appropriate for a Database and Filter node; if you create this code for a Compute node, use the correlation name InputRoot in place of Root.

```
IF Root.XMLNS.TestCase.Action = 'INSERT' THEN
   INSERT INTO Database.STOCK (STOCK_ID, STOCK_DESC, STOCK_QTY_HELD,
   BROKER_BUY_PRICE, BROKER_SELL_PRICE, STOCK_HIGH_PRICE, STOCK_HIGH_DATE,
   STOCK_HIGH_TIME) VALUES
   (CAST(Root.XMLNS.TestCase.stock_id AS INTEGER),
    Root.XMLNS.TestCase.stock_desc,
    CAST(Root.XMLNS.TestCase.stock_qty_held AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_buy_price AS DECIMAL),
    CAST(Root.XMLNS.TestCase.broker_sell_price AS DECIMAL),
    Root.XMLNS.TestCase.stock_high_price,
    CURRENT_DATE,
    CURRENT_TIME);

ELSEIF Root.XMLNS.TestCase.Action = 'DELETE' THEN

        DELETE FROM Database.STOCK WHERE STOCK.STOCK_ID =
                    CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);

   ELSEIF Root.XMLNS.TestCase.Action = 'UPDATE' THEN

           UPDATE Database.STOCK as A SET STOCK_DESC = Root.XMLNS.TestCase.stock_desc
                  WHERE  A.STOCK_ID = CAST(Root.XMLNS.TestCase.stock_id AS INTEGER);
END IF;
```

Checking returns to SELECT
If a SELECT function returns no data, or no further data, this result is handled as a normal situation and no error code is set in SQLCODE, regardless of the setting of the `Throw Exception On Database Error` and `Treat Warnings As Errors` properties on the current node.

About this task

To recognize that a SELECT function has returned no data, include ESQL that checks what has been returned. You can use various methods:

1. EXISTS

   This ESQL returns a Boolean value that indicates if a SELECT function returned one or more values (TRUE), or none (FALSE).

   ```
   IF EXISTS(SELECT T.MYCOL FROM Database.MYTABLE) THEN
   ...
   ```

2. CARDINALITY

   If you expect an array in response to a SELECT, you can use CARDINALITY to calculate how many entries have been received.

   ```
   SET OutputRoot.XMLNS.Testcase.Results[] = (
       SELECT T.MYCOL FROM Database.MYTABLE)
   ......
   IF CARDINALITY (OutputRoot.XMLNS.Testcase.Results[])> 0 THEN
   ........
   ```

3. IS NULL

   If you have used either THE or ITEM keywords in your SELECT function, a scalar value is returned. If no rows have been returned, the value set is NULL. However, it is possible that the value NULL is contained within the column, and you might want to distinguish between these two cases.

   Distinguish between cases by including COALESCE in the SELECT function, for example:

   ```
   SET OutputRoot.XMLNS.Testcase.Results VALUE = THE (
      SELECT ITEM COALESCE(T.MYCOL, 'WAS NULL')
      FROM Database.MYTABLE);
   ```

   If this example returns the character string WAS NULL, it indicates that the column contained NULL, and not that no rows were returned.

In previous releases, an SQLCODE of 100 was set in most cases if no data, or no further data, was returned. An exception was raised by the integration node if you chose to handle database errors in the message flow.

Committing database updates
When you create a message flow that interacts with databases, you can choose whether the updates that you make are committed when the current node has completed processing, or when the current invocation of the message flow has terminated.

About this task

The information in this topic does not apply to the DatabaseInput node. For more information about the DatabaseInput node, see "Event-based database integration" on page 1136.

You must use separate ODBC connections if you want to include nodes with `Automatic` transaction status and nodes with `Commit` transaction status in the same message flow, where the nodes operate on the same external database. Set up one connection for the nodes that are not to commit until the completion of the message flow, and a second connection for the nodes that are to commit immediately.

For each node, select the appropriate option for the `Transaction` property to specify when its database updates are to be committed:

- Choose `Automatic` (the default) if you want updates made in this node to be committed or rolled back as part of the whole message flow. The actions that you define in the ESQL module are performed on the message and it continues through the message flow. If the message flow completes successfully, the updates are committed. If the message flow fails, the message and the database updates are rolled back.
- Choose `Commit` if you want to commit the action of the node on the database, irrespective of the success or failure of the message flow as a whole. The database update is committed when the node processing is successfully completed, that is, after all ESQL has been processed, even if the message flow itself detects an error in a subsequent node that causes the message to be rolled back.

The value that you choose is implemented for the database tables that you have updated. You cannot select a different value for each table.

If you have set `Transaction` to `Commit`, the behavior of the message flow and the commitment of database updates can be affected by the use of the PROPAGATE statement in the node's ESQL.

If you choose to include a PROPAGATE statement that generates one or more output messages from the node, the processing of the PROPAGATE statement is not considered complete until the entire path that the output message takes has completed. This path might include several other nodes, including one or more output nodes. Only then does the node that issues the PROPAGATE statement receive control back and its ESQL terminate. At that point, a database commit is performed, if appropriate.

If one of the nodes on the propagated path detects an error and throws an exception, the processing of the node in which you have coded the PROPAGATE statement never completes. If the error processing results in a rollback, the message flow and the database update in this node are rolled back. This behavior is consistent with the stated operation of the `Commit` option, but might not be the behavior that you expect.

Invoking stored procedures
To invoke a procedure that is stored in a database, use the ESQL CALL statement. The stored procedure must be defined by a CREATE PROCEDURE statement that has a Language clause of DATABASE and an EXTERNAL NAME clause that identifies the name of the procedure in the database and, optionally, the database schema to which it belongs.

About this task

When you invoke a stored procedure with the CALL statement, the integration node ensures that the ESQL definition and the database definition match:

- The external name of the procedure must match a procedure in the database.

- The number of parameters must be the same.
- The type of each parameter must be the same.
- The direction of each parameter (IN, OUT, INOUT) must be the same.

The following restrictions apply to the use of stored procedures:

- Overloaded procedures are not supported. (An overloaded procedure is one that has the same name as another procedure in the same database schema with a different number of parameters, or parameters with different types.) If the integration node detects that a procedure is overloaded, it raises an exception.
- In an Oracle stored procedure declaration, you are not permitted to constrain CHAR and VARCHAR2 parameters with a length, and NUMBER parameters with a precision or scale, or both. Use %TYPE when you declare CHAR, VARCHAR, and NUMBER parameters to provide constraints on a formal parameter.
- Avoid changing the type or number of parameters of a procedure without also changing the name. If changes of this nature cannot be avoided, any integration servers with procedures with changed parameters called by the ESQL must be restarted.

Creating a stored procedure in ESQL

About this task

When you define an ESQL procedure that corresponds to a database stored procedure, you can specify either a qualified name (where the qualifier is a database schema) or an unqualified name.

To create a stored procedure:

Procedure

1. Code a statement similar to this example to create an unqualified procedure:

   ```
   CREATE PROCEDURE myProc1(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "myProc";
   ```

   The EXTERNAL NAME that you specify must match the definition that you have created in the database, but you can specify any name you choose for the corresponding ESQL procedure.
2. Code a statement similar to this example to create a qualified procedure:

   ```
   CREATE PROCEDURE myProc2(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL NAME "Schema1.myProc";
   ```
3. Code a statement similar to this example to create a qualified procedure in an Oracle package:

   ```
   CREATE PROCEDURE myProc3(IN p1 CHAR) LANGUAGE DATABASE EXTERNAL
                   NAME "mySchema.myPackage.myProc";
   ```

Results

For examples of stored procedure definitions in the database, see the CREATE PROCEDURE statement.

Calling a stored procedure

Procedure

1. Code a statement similar to this example to invoke an unqualified procedure:

   ```
   CALL myProc1('HelloWorld');
   ```

   Because it is not defined explicitly as belonging to any schema, the `myProc1` procedure must exist in the default schema (the name of which is the user name that is used to connect to the data source) or the command fails.

2. The following example calls the myProc procedure in schema Schema1.

```
CALL myProc2('HelloWorld');
```

3. Code a statement similar to this example to invoke an unqualified procedure with a dynamic schema:

```
DECLARE Schema2 char 'mySchema2';
CALL myProc1('HelloWorld') IN Database.{'Schema2'};
```

   This statement calls the myProc1 procedure in database Schema2, overriding the default "username" schema.

Calling a stored procedure that returns two result sets

About this task

To call a stored procedure that takes one input parameter and returns one output parameter and two result sets:

Procedure

1. Define the procedure with a CREATE PROCEDURE statement that specifies one input parameter, one output parameter, and two result sets:

```
CREATE PROCEDURE myProc1 (IN P1 INT, OUT P2 INT)
  LANGUAGE DATABASE
  DYNAMIC RESULT SETS 2
  EXTERNAL NAME "myschema.myproc1";
```

2. To invoke the myProc1 procedure by using a field reference, code:

```
/* using a field reference */
CALL myProc1(InVar1, OutVar2, Environment.ResultSet1[],
            OutputRoot.XMLNS.Test.ResultSet2[]);
```

3. To invoke the myProc1 procedure by using a reference variable, code:

```
/* using a reference variable*/
DECLARE cursor REFERENCE TO OutputRoot.XMLNS.Test;

CALL myProc1(InVar1, cursor.OutVar2, cursor.ResultSet1[],
                     cursor.ResultSet2[]);
```

Coding ESQL to handle errors

When you process messages in a message flow, errors can have a number of different causes and the message flow designer must decide how to handle those errors.

Introduction

When you process messages in message flows, errors can have the following causes:

- External causes; for example, the incoming message is syntactically invalid, a database used by the flow has been shut down, or the power supply to the machine on which the integration node is running fails.
- Internal causes; for example, an attempt to insert a row into a database table fails because of a constraint check, or a character string that is read from a database cannot be converted to a number because it contains alphabetic characters.

  Internal errors can be caused by programs storing invalid data in the database, or by a flaw in the logic of a flow.

The message flow designer must decide how to handle errors.

Using default error-handling

The simplest strategy for handling ESQL errors is to do nothing, and use the integration node's default behavior. The default behavior is to cut short the processing of the failing message, and to proceed to the next message. Input and output nodes provide options to control exactly what happens when processing is cut short.

If the input and output nodes are set to transactional mode, the integration node restores the state before the message is processed:

1. The input message that has apparently been taken from the input queue is put back.
2. Any output messages that the flow has apparently written to output queues are discarded.

If the input and output nodes are not set to transactional mode:

1. The input message that was taken from the input queue is not put back.
2. Any output messages that the flow has written to output queues remain on the output queues.

Each of these strategies has its advantages. The transactional model preserves the consistency of data, while the non-transactional model maximizes the continuity of message processing. In the transactional model, the failing input message is put back onto the input queue, and the integration node attempts to process it again. The most likely outcome of this scenario is that the message continues to fail until the retry limit is reached, at which point the message is placed on a dead letter queue. The reason for the failure to process the message is logged to the system event log (Windows) or syslog (UNIX). Therefore, the failing message holds up the processing of subsequent valid messages, and is left unprocessed by the integration node.

Most databases operate transactionally so that all changes that are made to database tables are committed if the processing of the message succeeds, or rolled back if it fails, therefore maintaining the integrity of data. An exception to this situation is if the integration node itself, or a database, fails (for example, the power to the computers on which they are running is interrupted). In these cases, changes might be committed in some databases, but not in others, or the database changes might be committed but the input and output messages are not committed. If these possibilities concern you, make the flow coordinated and configure the databases that are involved.

Using customized error handling

The following list contains some general tips for creating customized error handlers.

- If you require something better than default error handling, the first step is to use a handler; see DECLARE HANDLER statement. Create one handler per node to intercept all possible exceptions (or all those that you can predict).
- Having intercepted an error, the error handler can use whatever logic is appropriate to handle it. Alternatively, it can use a THROW statement or node to create an exception, which could be handled higher in the flow logic, or even reach the input node, causing the transaction to be rolled back; see “Throwing an exception” on page 1680.
- If a node generates an exception that is not caught by the handler, the flow is diverted to the Failure terminal, if one is connected, or handled by default error-handling if no Failure terminal is connected.

  Use Failure terminals to catch unhandled errors. Attach a simple logic flow to the Failure terminal. This logic flow could consist of a database or Compute node that writes a log record to a database (possibly including the message's bit stream), or writes a record to the event log. The flow could also contain an output node that writes the message to a special queue.

  The full exception tree is passed to any node that is connected to a Failure terminal; see “Exception list tree” on page 516.
- Your error handlers are responsible for logging each error in an appropriate place, such as the system event log.

For a detailed description of the options that you can use to process errors in a message flow, see “Handling errors in message flows” on page 1883. For examples of what you can do, see “Throwing an exception” on page 1680 and “Capturing database state” on page 1680.

Writing code to detect errors

The following sections assume that the integration node detects the error. It is possible, however, for the logic of the flow to detect an error. For example, when coding the flow logic, you could use the following elements:

- IF statements that are inserted specifically to detect situations that should not occur
- The ELSE clause of a case expression or statement to trap routes through the code that should not be possible

As an example of a flow logic-detected error, consider a field that has a range of possible integer values that indicate the type of message. It would not be good practice to leave to chance what would happen if a message were to arrive in which the field's value did not correspond to any known type of message. One way this situation could occur is if the system is upgraded to support extra types of message, but one part of the system is not upgraded.

Using your own logic to handle input messages that are not valid

Input messages that are syntactically invalid (and input messages that appear to be not valid because of erroneous message format information) are difficult to deal with, because the integration node cannot determine the contents of the message. Typically, the best way to deal with these messages is to configure the input node to fully parse and validate the message. However, this configuration applies only to predefined messages; that is, MRM or IDoc.

If the input node is configured in this way, the following results are guaranteed if the input message cannot be parsed successfully:

- The input message never emerges from the node's normal output terminal (it goes to the Failure terminal).
- The input message never enters the main part of the message flow.
- The input message never causes any database updates.
- No messages are written to any output queues.

To deal with a failing message, connect a simple logic flow to the Failure terminal. The only disadvantage to this strategy is that if the normal flow does not require access to all of the message's fields, the forcing of complete parsing of the message affects performance.

Using your own logic to handle database errors

Database errors fall into three categories:

- The database is not working (for example, it is off line).
- The database is working but refuses your request (for example, a lock contention occurs).
- The database is working but it cannot do what you request (for example, read from a non-existent table).

If you require something better than default error handling, the first step is to use a handler (see DECLARE HANDLER statement) to intercept the exception. The handler can determine the nature of the failure from the SQL state that is returned by the database.

A database is not working

If a database is not working at all, and is essential to the processing of messages, there is typically not much that you can do. The handler, having determined the cause, might complete one or more of the following actions:

- Use the RESIGNAL statement to re-throw the original error, therefore allowing the default error handler to take over
- Use a different database

- Write the message to a special output queue

  Be careful if you use an approach similar to this technique; the handler absorbs the exception, therefore all changes to other databases, or writes to queues, are committed.

A database refuses your request
The situation when a lock contention occurs is similar to the "Database not working" case because the database will have backed out *all* the database changes that you have made for the current message, not just the failing request. Therefore, unless you are sure that this was the only update, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Impossible requests
An impossible request occurs when the database is working, but cannot complete the requested action. This type of error covers a wide variety of problems.

If, as discussed in the previous example, the database does not have a table of the name that the flow expects, default error-handling is typically the best strategy, except possibly logging the error or passing the message to a special queue.

Many other errors might be handled successfully, however. For example, an attempt to insert a row might fail because there is already such a row and the new row would be a duplicate. Or an attempt to update a row might fail because there is no such row (that is, the update action updated zero rows). In these cases, the handler can incorporate whatever logic you think appropriate. It might insert the missing row, or use the existing one (possibly making sure the values in it are suitable).

If you want an update of zero rows to be reported as an error, you must set the `Treat warnings as errors` property on the node to `true`, which is not the default setting.

Using your own logic to handle errors in output nodes

Errors that occur in MQOutput nodes report the nature of the error in the SQL state and give additional information in the *SQL native error* variable. Therefore, if something better than default error handling is required, the first step is to use a handler (see DECLARE HANDLER statement) to intercept the exception. Such a handler typically surrounds only a single PROPAGATE statement.

Using your own logic to handle other errors

Besides those errors covered above, a variety of other errors can occur. For example, an arithmetic calculation might overflow, a cast might fail because of the unsuitability of the data, or an access to a message field might fail because of a type constraint. The integration node offers two programming strategies for dealing with these types of error.

- The error causes an exception that is either handled or left to roll back the transaction.
- The failure is recorded as a special value that is tested for later.

In the absence of a type constraint, an attempt to access a non-existent message field results in the value null. Null values propagate through expressions, making the result null. Therefore, if an expression, however complex, does not return a null value, you know that all the values that it needed to calculate its result were not null.

Cast expressions can have a default clause. If there is a default clause, casts fail quietly; instead of throwing an exception, they simply return the default value. The default value could be an innocuous number (for example, zero for an integer), or a value that is clearly invalid in the context (for example, -1 for a customer number). Null might be particularly suitable because it is a value that is different from all others, and it will propagate through expressions without any possibility of the error condition being masked.

Handling errors in other nodes

Exceptions that occur in other nodes (that is, downstream of a PROPAGATE statement) might be caught by handlers in the normal way. Handling such errors intelligently, however, poses a problem: another

node was involved in the original error, therefore another node, and not necessarily the originator of the exception, is likely to be involved in handling the error.

To help in these situations, the Database and Compute nodes have four terminals called Out1, Out2, Out3, and Out4. In addition, the syntax of the PROPAGATE statement includes target expression, message source, and control clauses to give more control over these terminals.

Throwing an exception
If you detect an error or other situation in your message flow in which you want message processing to be ended, you can throw an exception in a message flow in two ways.

About this task

- Use the ESQL THROW EXCEPTION statement.

  Include the THROW statement anywhere in the ESQL module for a Compute, Database, or Filter node. Use the options on the statement to code your own data to be inserted into the exception.
- Include a THROW node in your message flow.

  Set the node properties to identify the source and content of the exception.

By using either statement options or node properties, you can specify a message identifier and values that are inserted into the message text to give additional information and identification to users who interpret the exception. You can specify any message in any catalog that is available to the integration server.

The situations in which you might want to throw an exception are determined by the behavior of the message flow; decide when you design the message flow where this action might be appropriate. For example, you might want to examine the content of the input message to ensure that it meets criteria that cannot be detected by the input node (which might check that a particular message format is received).

The following example uses the Example message to show how you can use the ESQL THROW statement. To check that the invoice number is within a particular range, throw an exception for any invoice message received that does not fall in the valid range.

```
--Check for invoice number lower than permitted range
IF Body.Invoice.InvoiceNo < 100000 THEN
   THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1234 VALUES
   ('Invoice number too low', Body.Invoice.InvoiceNo);

-- Check for invoice number higher than permitted range
ELSEIF Body.Invoice.InvoiceNo > 500000 THEN
      THROW USER EXCEPTION CATALOG 'MyCatalog' MESSAGE 1235 VALUES
   ('Invoice number too high', Body.Invoice.InvoiceNo);

ELSE DO
  -- invoice number is within permitted range
  -- complete normal processing
ENDIF;
```

Capturing database state
If an error occurs when the integration node accesses an external database, you can either let the integration node throw an exception during message flow node processing or use ESQL statements to process the exception within the message flow node itself.

About this task

Letting the integration node throw an exception during message flow node processing is the default action; ESQL processing in the current message flow node is abandoned. The exception is then propagated backwards through the message flow until an enclosing catch node, or the input node for this message flow, is reached. If the exception reaches the input node, an active transaction is rolled back.

Using ESQL statements to process the exception within the message flow node itself requires an understanding of database return codes and a logical course of action to take when an error occurs. To enable this inline database error processing, you must clear the Filter, Database, or Compute node's `Throw Exception On Database Error` property. If you clear this property, the message flow node

sets the database state indicators SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT, with appropriate information from the database manager instead of throwing an exception.

The indicators contain information only when an error (not a warning) occurs, unless you have selected the `Treat Warnings As Errors` property. If a database operation is successful, or returns success with information, the indicators contain their default success values.

You can use the values contained in these indicators in ESQL statements to make decisions about the action to take. You can access these indicators with the SQLCODE, SQLSTATE, SQLNATIVEERROR, and SQLERRORTEXT functions.

If you are attempting inline error processing, check the state indicators after each database statement is executed to ensure that you catch and assess all errors. When processing the indicators, if you meet an error that you cannot handle inline, you can raise a new exception either to deal with it upstream in a catch node, or to let it through to the input node so that the transaction is rolled back, for which you can use the ESQL THROW statement.

You might want to check for the special case in which a SELECT returns no data. This situation is not considered an error and SQLCODE is not set, therefore you must test explicitly for it; see "Checking returns to SELECT" on page 1673.

Using ESQL to access database state indicators

The following ESQL example shows how to use the four database state functions, and how to include the error information that is returned in an exception:

```
DECLARE SQLState1 CHARACTER;
DECLARE SQLErrorText1 CHARACTER;
DECLARE SQLCode1 INTEGER;
DECLARE SQLNativeError1 INTEGER;

-- Make a database insert to a table that does not exist --
INSERT INTO Database.DB2ADMIN.NONEXISTENTTABLE (KEY,QMGR,QNAME)
                              VALUES (45,'REG356','my TESTING 2');

--Retrieve the database return codes --
SET SQLState1 = SQLSTATE;
SET SQLCode1 = SQLCODE;
SET SQLErrorText1 = SQLERRORTEXT;
SET SQLNativeError1 = SQLNATIVEERROR;

--Use the THROW statement to back out the database and issue a user exception--
THROW USER EXCEPTION MESSAGE 2950 VALUES
( 'The SQL State' , SQLState1 , SQLCode1 , SQLNativeError1 ,
SQLErrorText1 );
```

You do not have to throw an exception when you detect a database error; you might prefer to save the error information returned in the local environment tree, and include a Filter node in your message flow that routes the message to error or success subflows according to the values saved.

Using the SELECT function
The SELECT function is a convenient and powerful tool for accessing fields and transforming data in a message tree.

About this task

The following topics show by example how to use the SELECT function to transform a variety of messages. The examples are based on an XML input message that has been parsed in the XMLNS domain. However, the techniques shown in these topics can be applied to any message tree.

- "Transforming a simple message" on page 1682
- "Transforming a complex message" on page 1684
- "Returning a scalar value in a message" on page 1686
- "Joining data in a message" on page 1688
- "Translating data in a message" on page 1689

- "Joining data from messages and database tables" on page 1691

Transforming a simple message
When you code the ESQL for a Compute node, use the SELECT function to transform simple messages.

About this task

This topic provides examples of simple message transformation. Review the examples and modify them for your own use. They are all based on the Example message as input.

Example

Consider the following ESQL:

```
 SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity, R.Author FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

When this ESQL code processes the Invoice message, it produces the following output message:

```
<Data>
  <Output>
     <Quantity>2</Quantity>
     <Author>Neil Bradley</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Don Chamberlin</Author>
  </Output>
  <Output>
     <Quantity>1</Quantity>
     <Author>Philip Heller, Simon Roberts</Author>
  </Output>
</Data>
```

Three Output fields are present, one for each Item field, because SELECT creates an item in its result list for each item described by its FROM list. Within each Output field, a Field is created for each field named in the SELECT clause. These fields are in the order in which they are specified within the SELECT clause, not in the order in which they appear in the incoming message.

The R that is introduced by the final AS keyword is known as a correlation name. It is a local variable that represents in turn each of the fields addressed by the FROM clause. The name chosen has no significance. In summary, this simple transform does two things:

1. It discards unwanted fields.
2. It guarantees the order of the fields.

You can perform the same transform with a procedural algorithm:

```
DECLARE i INTEGER 1;
DECLARE count INTEGER CARDINALITY(InputRoot.XMLNS.Invoice.Purchases.Item[]);

WHILE (i <= count)
   SET OutputRoot.XMLNS.Data.Output[i].Quantity = InputRoot.XMLNS.Invoice.Purchases.Item[i].Quantity;
   SET OutputRoot.XMLNS.Data.Output[i].Author   = InputRoot.XMLNS.Invoice.Purchases.Item[i].Author;
   SET i = i+1;
END WHILE;
```

These examples show that the SELECT version of the transform is much more concise. It also executes faster.

The following example shows a more advanced transformation:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

In this transform, an AS clause is associated with each item in the SELECT clause. This clause gives each field in the result an explicit name rather than a field name that is inherited from the input. These names can be paths (that is, a dot-separated list of names), as shown in the example. The structure of the output message structure can be different from the input message. Using the same Invoice message, the result is:

```
<Data>
  <Output>
    <Book>
      <Quantity>2</Quantity>
      <Author>Neil Bradley</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Don Chamberlin</Author>
    </Book>
  </Output>
  <Output>
    <Book>
      <Quantity>1</Quantity>
      <Author>Philip Heller, Simon Roberts</Author>
    </Book>
  </Output>
</Data>
```

The expressions in the SELECT clause can be of any complexity and there are no special restrictions. They can include operators, functions, and literals, and they can refer to variables or fields that are not related to the correlation name. The following example shows more complex expressions:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT 'Start'                          AS Header,
            'Number of books:' || R.Quantity AS Book.Quantity,
            R.Author || ':Name and Surname'  AS Book.Author,
            'End'                            AS Trailer
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result in this case is:

```
<Data>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:2</Quantity>
   <Author>Neil Bradley:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Don Chamberlin:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
 <Output>
  <Header>Start</Header>
  <Book>
   <Quantity>Number of books:1</Quantity>
   <Author>Philip Heller, Simon Roberts:Name and Surname</Author>
  </Book>
  <Trailer>End</Trailer>
 </Output>
</Data>
```

As shown above, the AS clauses of the SELECT clause contain a path that describes the full name of the field that is to be created in the result. These paths can also specify (as is normal for paths) the type of field that is to be created. The following example transform specifies the field types. In this case, XML tagged data is transformed to XML attributes:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity.* AS Book.(XML.Attribute)Quantity,
            R.Author.*   AS Book.(XML.Attribute)Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
    );
```

Using the same Invoice message, the result is:

```
<Data>
 <Output>
  <Book Quantity="2" Author="Neil Bradley"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Don Chamberlin"/>
 </Output>
 <Output>
  <Book Quantity="1" Author="Philip Heller, Simon Roberts"/>
 </Output>
</Data>
```

Finally, you can use a WHERE clause to eliminate some of the results. In the following example a WHERE clause is used to remove results in which a specific criterion is met. An entire result is either included or excluded:

```
SET OutputRoot.XMLNS.Data.Output[] =
    (SELECT R.Quantity AS Book.Quantity,
            R.Author   AS Book.Author
            FROM InputRoot.XMLNS.Invoice.Purchases.Item[] AS R
            WHERE R.Quantity = 2
    );
```

Using the same input message, the result is:

```
<Data>
 <Output>
  <Book>
   <Quantity>2</Quantity>
   <Author>Neil Bradley</Author>
  </Book>
 </Output>
</Data>
```

Transforming a complex message
When you code the ESQL for a Compute node, use the SELECT function for complex message transformation.

About this task

This topic provides examples of complex message transformation. Review the examples and modify them for your own use. They are all based on the Example message as input.

Example

In this example, Invoice contains a variable number of Items. The transform is shown in the following example:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                    AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName  AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')                   AS Customer.Phone,
            (SELECT II.Title                          AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )     AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                          )
             FROM I.Purchases.Item[] AS II      )       AS Amount,
            'Dollars'                                   AS Amount.(XML.Attribute)Currency

            FROM InputRoot.XMLNS.Invoice[] AS I
            WHERE I.Customer.LastName <> 'Brown'
    );
```

The output message that is generated is:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Paperback" Edition="2">
          A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc Category="Computer" Form="Hardcover" Edition="0">JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

This transform has nested SELECT clauses. The outer statement operates on the list of Invoices. The inner statement operates on the list of Items. The AS clause that is associated with the inner SELECT clause expects an array:

```
            (SELECT II.Title                          AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             )
            -- Note the use of [] in the next expression
               AS Purchases.Article[],
```

This statement tells the outer SELECT clause to expect a variable number of Items in each result. Each SELECT clause has its own correlation name: I for the outer SELECT clause and II for the inner one. Each SELECT clause typically uses its own correlation name, but the FROM clause in the inner SELECT clause refers to the correlation name of the outer SELECT clause:

```
            (SELECT II.Title                          AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                       AS Qty
            -- Note the use of I.Purchases.Item in the next expression
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0             ) AS Purchases.Article[],
```

This statement tells the inner SELECT clause to work with the current Invoice's Items. Both SELECT clauses contain WHERE clauses. The outer one uses one criterion to discard certain Customers, and the inner one uses a different criterion to discard certain Items. The example also shows the use of COALESCE to prevent missing input fields from causing the corresponding output field to be missing. Finally, it also uses the column function SUM to add together the value of all Items in each Invoice. Column functions are discussed in “Referencing columns in a database” on page 1666.

When the fields Desc are created, the whole of the input Title field is copied: the XML attributes and the field value. If you do not want these attributes in the output message, use the FIELDVALUE function to discard them; for example, code the following ESQL:

```
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.Customer.Title                                AS Customer.Title,
            I.Customer.FirstName || ' ' || I.Customer.LastName AS Customer.Name,
            COALESCE(I.Customer.PhoneHome,'')               AS Customer.Phone,
            (SELECT FIELDVALUE(II.Title)                    AS Desc,
                    CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                    II.Quantity                         AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE  II.UnitPrice> 0.0              )      AS Purchases.Article[],
            (SELECT SUM( CAST(II.UnitPrice AS FLOAT) *
                    CAST(II.Quantity  AS FLOAT) *
                    1.6                           )
             FROM I.Purchases.Item[] AS II        ) AS Amount,
            'Dollars'                               AS Amount.(XML.Attribute)Currency

    FROM InputRoot.XMLNS.Invoice[] AS I
    WHERE I.Customer.LastName <> 'Brown'
    );
```

That code generates the following output message:

```
<Data>
 <Statement>
  <Customer>
   <Title>Mr</Title>
   <Name>Andrew Smith</Name>
   <Phone>01962818000</Phone>
  </Customer>
  <Purchases>
   <Article>
    <Desc>The XML Companion</Desc>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Desc>A Complete Guide to DB2 Universal Database</Desc>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Desc>JAVA 2 Developers Handbook</Desc>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
 </Statement>
</Data>
```

Returning a scalar value in a message
Use a SELECT statement to return a scalar value by including both the **THE** and **ITEM** keywords.

About this task

For example:

```
1 + THE(SELECT ITEM T.a FROM Body.Test.A[] AS T WHERE T.b = '123')
```

Use of the ITEM keyword

About this task

The following example shows the use of the ITEM keyword to select one item and create a single value.

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result[] =
   (SELECT ITEM T.UnitPrice FROM InputBody.Invoice.Purchases.Item[] AS T);
```

When the Invoice message is received as input, the ESQL shown generates the following output message:

```
<Test>
  <Result>27.95</Result>
  <Result>42.95</Result>
  <Result>59.99</Result>
</Test>
```

When the ITEM keyword is specified, the output message includes a list of scalar values. Compare this message to the one that is produced if the ITEM keyword is omitted, in which a list of fields (name-value pairs) is generated:

```
<Test>
  <Result>
    <UnitPrice>27.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>42.95</UnitPrice>
  </Result>
  <Result>
    <UnitPrice>59.99</UnitPrice>
  </Result>
</Test>
```

Effects of the THE keyword

About this task

The THE keyword converts a list containing one item to the item itself.

The two previous examples both specified a list as the source of the SELECT in the FROM clause (the field reference has [] at the end to indicate an array), so typically the SELECT function generates a list of results. Because of this behavior, you must specify a list as the target of the assignment (thus the "Result[]" as the target of the assignment). However, you often know that the WHERE clause that you specify as part of the SELECT returns TRUE for only one item in the list. In this case use the THE keyword.

The following example shows the effect of using the THE keyword:

```
SET OutputRoot.MQMD = InputRoot.MQMD;

SET OutputRoot.XMLNS.Test.Result =
    THE (SELECT T.Publisher, T.Author FROM InputBody.Invoice.Purchases.Item[]
         AS T WHERE T.UnitPrice = 42.95);
```

The THE keyword means that the target of the assignment becomes `OutputRoot.XMLNS.Test.Result` (the "[]" is not permitted). Its use generates the following output message:

```
<Test>
  <Result>
    <Publisher>Morgan Kaufmann Publishers</Publisher>
    <Author>Don Chamberlin</Author>
  </Result>
</Test>
```

Selecting from a list of scalars

About this task

Consider the following sample input message:

```
<Test>
 <A>1</A>
 <A>2</A>
 <A>3</A>
 <A>4</A>
 <A>5</A>
</Test>
```

If you code the following ESQL statements to process this message:

```
SET OutputRoot.XMLNS.Test.A[] =
  (SELECT ITEM A from InputBody.Test.A[]
   WHERE CAST(A AS INTEGER) BETWEEN 2 AND 4);
```

the following output message is generated:

```
      <A>2</A>
      <A>3</A>
      <A>4</A>
```

Joining data in a message

The FROM clause of a SELECT function is not restricted to having one item. Specifying multiple items in the FROM clause produces the typical Cartesian product joining effect, in which the result includes an item for all combinations of items in the two lists.

About this task

Using the FROM clause in this way produces the same joining effect as standard SQL.

The Invoice message includes a set of customer details, payment details, and details of the purchases that the customer makes. Code the following ESQL to process the input Example message:

```
SET OutputRoot.XMLNS.Items.Item[] =
   (SELECT D.LastName, D.Billing,
           P.UnitPrice, P.Quantity
    FROM InputBody.Invoice.Customer[] AS D,
         InputBody.Invoice.Purchases.Item[] AS P);
```

The following output message is generated:

```
<Items>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>27.95</UnitPrice>
  <Quantity>2</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>42.95</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
 <Item>
  <LastName>Smith</LastName>
  <Billing>
   <Address>14 High Street</Address>
   <Address>Hursley Village</Address>
   <Address>Hampshire</Address>
   <PostCode>SO213JR</PostCode>
  </Billing>
  <UnitPrice>59.99</UnitPrice>
  <Quantity>1</Quantity>
 </Item>
</Items>
```

Three results are produced, giving the number of descriptions in the first list (one) multiplied by the number of prices in the second (three). The results systematically work through all the combinations of the two lists. You can see this by looking at the *LastName* and *UnitPrice* fields selected from each result:

```
LastName Smith   UnitPrice 27.95
LastName Smith   UnitPrice 42.95
LastName Smith   UnitPrice 59.99
```

You can join data that occurs in a list and a non-list, or in two non-lists, and so on. For example:

```
OutputRoot.XMLNS.Test.Result1[] =
  (SELECT ... FROM InputBody.Test.A[], InputBody.Test.b);
OutputRoot.XMLNS.Test.Result1 =
  (SELECT ... FROM InputBody.Test.A, InputBody.Test.b);
```

The location of the [] in each case is significant. Any number of items can be specified in the FROM clause, not just one or two. If any of the items specify [] to indicate a list of items, the SELECT function returns a list of results (the list might contain only one item, but the SELECT function can return a list of items).

The target of the assignment must specify a list (so must end in []), or you must use the THE function if you know that the WHERE clause guarantees that only one combination is matched.

Translating data in a message
You can translate data from one form to another.

About this task

A typical example of the requirement to translate data is if the items are known in one message by names, and in another message by numbers. For example:

```
Type Name          Type Code

Confectionary      2000
Newspapers         3000
Hardware           4000
```

Consider the following input message:

```
<Data>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>Confectionary</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>NewsPapers</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>Hardware</Type>
    </Item>
  </Items>
  <TranslateTable>
    <Translate>
      <Name>Confectionary</Name>
      <Number>2000</Number>
    </Translate>
    <Translate>
      <Name>NewsPapers</Name>
      <Number>3000</Number>
    </Translate>
    <Translate>
      <Name>Hardware</Name>
      <Number>4000</Number>
    </Translate>
  </TranslateTable>
</Data>
```

This message has two sections; the first section is a list of items in which each item has a catalog number and a type; the second section is a table for translating between descriptive type names and numeric type codes. Include a Compute node with the following transform:

```
SET OutputRoot.XMLNS.Result.Items.Item[] =
   (SELECT M.Cat, M.Description, T.Number As Type
     FROM
       InputRoot.XMLNS.Data.Items.Item[]               As M,
       InputRoot.XMLNS.Data.TranslateTable.Translate[] As T
     WHERE M.Type = T.Name
   );
```

The following output message is generated:

```
<Result>
  <Items>
    <Item>
      <Cat>1000</Cat>
      <Description>Milk Chocolate Bar</Description>
      <Type>2000</Type>
    </Item>
    <Item>
      <Cat>1001</Cat>
      <Description>Daily Newspaper</Description>
      <Type>3000</Type>
    </Item>
    <Item>
      <Cat>1002</Cat>
      <Description>Kitchen Sink</Description>
      <Type>4000</Type>
    </Item>
  </Items>
</Result>
```

In the result, each type name has been converted to its corresponding code. In this example, both the data and the translate table were in the same message tree, although this is not a requirement. For example, the translate table could be coded in a database, or might have been set up in LocalEnvironment by a previous Compute node.

Joining data from messages and database tables
You can use SELECT functions that interact with both message data and databases.

About this task

You can also nest a SELECT function that interacts with one type of data within a SELECT clause that interacts with the other type.

Consider the following input message, which contains invoice information for two customers:

```
<Data>
  <Invoice>
    <CustomerNumber>1234</CustomerNumber>
    <Item>
      <PartNumber>1</PartNumber>
      <Quantity>9876</Quantity>
    </Item>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>8765</Quantity>
    </Item>
</Invoice>
  <Invoice>
    <CustomerNumber>2345</CustomerNumber>
    <Item>
      <PartNumber>2</PartNumber>
      <Quantity>7654</Quantity>
    </Item>
    <Item>
      <PartNumber>1</PartNumber>
    <Quantity>6543</Quantity>
    </Item>
</Invoice>
</Data>
```

Consider the following database tables, Prices and Addresses, and their contents:

```
PARTNO      PRICE
----------- ----------------------
          1            +2.50000E+001
          2            +6.50000E+00



PARTNO      STREET                CITY             COUNTRY
------      ------------------    --------------   -------
1234        22 Railway Cuttings   East Cheam       England
2345        The Warren            Watership Down   England
```

If you code the following ESQL transform:

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
   (SELECT I.CustomerNumber                          AS Customer.Number,
           A.Street                                  AS Customer.Street,
           A.City                                    AS Customer.Town,
           A.Country                                 AS Customer.Country,

        -- Select suitable items
          (SELECT II.PartNumber AS PartNumber,
                  II.Quantity   AS Quantity,
                  PI.Price      AS Price
           FROM Database.db2admin.Prices AS PI,
               I.Item[]                  AS II
           WHERE II.PartNumber = PI.PartNo     )    AS Purchases.Item[]

    FROM Database.db2admin.Addresses  AS A,
         InputRoot.XMLNS.Data.Invoice[] AS I

    WHERE I.CustomerNumber = A.PartNo
    );
```

the following output message is generated. The input message is augmented with the price and address information from the database table:

```
<Data>
  <Statement>
    <Customer>
      <Number>1234</Number>
      <Street>22 Railway Cuttings</Street>
      <Town>East Cheam</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>9876</Quantity>
        <Price>2.5E+1</Price>
      </Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>8765</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
  <Statement>
    <Customer>
      <Number>2345</Number>
      <Street>The Warren</Street>
      <Town>Watership Down</Town>
      <Country>England</Country>
    </Customer>
    <Purchases>
      <Item>
        <PartNumber>1</PartNumber>
        <Quantity>6543</Quantity>
        <Price>2.5E+1</Price></Item>
      <Item>
        <PartNumber>2</PartNumber>
        <Quantity>7654</Quantity>
        <Price>6.5E+1</Price>
      </Item>
    </Purchases>
  </Statement>
</Data>
```

You can nest the database SELECT clause within the message SELECT clause. In most cases, the code is not as efficient as the previous example, but you might find that it is better if the messages are small and the database tables are large.

```
-- Create a valid output message
SET OutputRoot.MQMD = InputRoot.MQMD;

-- Select suitable invoices
SET OutputRoot.XMLNS.Data.Statement[] =
    (SELECT I.CustomerNumber                  AS Customer.Number,

        -- Look up the address
        THE ( SELECT
                  A.Street,
                  A.City    AS Town,
                  A.Country
              FROM Database.db2admin.Addresses AS A
              WHERE A.PartNo = I.CustomerNumber
            )                              AS Customer,

        -- Select suitable items
        (SELECT
            II.PartNumber AS PartNumber,
            II.Quantity   AS Quantity,

            -- Look up the price
            THE (SELECT ITEM P.Price
              FROM Database.db2admin.Prices AS P
              WHERE P.PartNo = II.PartNumber
            )             AS Price

          FROM I.Item[] AS II           ) AS Purchases.Item[]

    FROM InputRoot.XMLNS.Data.Invoice[] AS I
    );
```

Manipulating messages in the DFDL domain
How to use messages that have been modeled by using DFDL schema files, and that are parsed by the DFDL parser.

About this task

The topics in this section provide information about how to write ESQL for processing messages that belong to the DFDL domain, and that are parsed by the DFDL parser. For more information about the DFDL parser and domain, see “DFDL parser and domain” on page 544.

The structure of the message tree that the DFDL parser builds, and the field types that it uses, are defined by the logical model that is defined in your DFDL schema file. For more information about DFDL schema files, see “Working with DFDL schema files” on page 2220. The following topics show you how to deal with messages that have been modeled in the DFDL domain, and that are parsed by the DFDL parser. Use this information in conjunction with the information about manipulating message body content; see “Manipulating message body content” on page 1632.

- Accessing elements in a message
- Accessing multiple occurrences of an element
- Accessing elements within groups
- Accessing elements that have namespace support enabled
- Querying null values
- Setting null values
- Working with bit streams

Accessing elements in a message in the DFDL domain
You can use ESQL to manipulate the logical tree that represents a message in your message flow. This topic describes how to access data for elements that are in a message in the DFDL domain.

About this task

You can populate an element with data by using the SET statement:

```
SET OutputRoot.DFDL.MyMessage.Name = UPPER(InputRoot.DFDL.MyMessage.Name);
```

The field references in the ESQL statement both refer to the element *Name* within the message *MyMessage*, that is modeled in a DFDL schema file. This statement takes the input value for the Name field, converts it to uppercase, and assigns the result to the same element in the output message.

Accessing multiple occurrences of an element in a message in the DFDL domain
You can use specific ESQL code to set the value of one occurrence of an element that has multiple occurrences in a message. You can also use arrow notation to indicate the direction of search when searching for multiple occurrences of an element.

About this task

You can access DFDL domain elements by following the general guidance given in "Accessing known multiple occurrences of an element" on page 1636 and "Accessing unknown multiple occurrences of an element" on page 1637. Further information that is specific to messages in the DFDL domain is provided in this topic.

Consider the following statements:

```
SET OutputRoot.DFDL.MyMessage.Loaned[1].Currency = 'GBP';
SET OutputRoot.DFDL.MyMessage.Loaned[2].Currency = 'USD';
```

The above SET statements operate on two occurrences of the element Loaned. Each statement sets the value of the child Currency. The array index indicates which occurrence of the repeating element you are interested in.

When you define child elements of a complex type in a message set, you can add the same element to the complex type more than once. These instances do not have to be contiguous, but you must use the same method (array notation) to refer to them in ESQL.

For example, if you create a complex type with a `Composition` of `Sequence` that contains the following elements:

- `StringElement1`
- `IntegerElement1`
- `StringElement1`

use the following ESQL to set the value of `StringElement1`:

```
SET OutputRoot.DFDL.MyMessage.StringElement1[1] =
              'This is the first occurrence of StringElement1';
SET OutputRoot.DFDL.MyMessage.StringElement1[2] =
              'This is the second occurrence of StringElement1';
```

You can also use the arrow notation (the greater than (>) and less than (<) symbols) to indicate the direction of search and the index to be specified:

```
SET OutputRoot.DFDL.MyMessage.StringElement1[>] =
               'This is the first occurrence of StringElement1';
SET OutputRoot.DFDL.MyMessage.StringElement1[<2] =
               'This is the last but one occurrence of
 StringElement1';
SET OutputRoot.DFDL.MyMessage.StringElement1[<1] =
               'This is the last occurrence of StringElement1';
```

Refer to "Accessing known multiple occurrences of an element" on page 1636 and "Accessing unknown multiple occurrences of an element" on page 1637 for additional detail.

Accessing elements within groups in a message in the DFDL domain
When an input message is parsed, structures that you have defined as local groups (sequences or choices) in your DFDL schema file are not directly represented in the logical tree, but their child elements are.

About this task

Local groups do not have names in the model, and so do not cause elements to be created in the logical message tree. If you want to refer to the values of elements that are children of a local group, do not include the group in the ESQL path.

Accessing the content of a message in the DFDL domain with namespace support enabled
Use namespaces where appropriate for messages that are parsed by the DFDL parser.

About this task

It is possible to create a DFDL schema file that has a target namespace. All messages defined in that DFDL schema belong to that namespace. When the DFDL parser parses one of the messages, the namespace is included in the elements that are created in the logical message tree. You must include the namespace when you code the ESQL reference to the element. If you do not include the namespace, the integration node searches the `no target` namespace. If the element is not found in the `no target` namespace, the integration node searches all other known namespaces in the message model schema file. For performance and integrity reasons, specify namespaces wherever they apply.

The most efficient way to refer to elements when namespaces are enabled is to define a namespace constant, and use this constant in the appropriate ESQL statements. This technique makes your ESQL code much easier to read and maintain.

Define a constant using the DECLARE NAMESPACE statement:

```
DECLARE ns01 NAMESPACE 'http://www.ns01.com'

SET OutputRoot.DFDL.ns01:MyMessage.ns01:Element1 = InputBody.ns01:MyMessage.ns01:Element1;
```

`ns01` is interpreted correctly as a namespace because of the way that it is declared.

You can also use a CHARACTER variable to declare a namespace:

```
DECLARE ns02 CHARACTER 'http://www.ns02.com'

SET OutputRoot.DFDL.{ns02}:MyMessage.{ns02}:Element1 = InputBody.{ns02}:MyMessage.
{ns02}:Element1;
```

If you use this method, you must surround the declared variable with braces to ensure that it is interpreted as a namespace.

If you are concerned that a CHARACTER variable might get changed, you can use a CONSTANT CHARACTER declaration:

```
DECLARE ns03 CONSTANT CHARACTER 'http://www.ns03.com'

SET OutputRoot.DFDL.{ns03}:MyMessage.{ns03}:Element1 = InputBody.{ns03}:MyMessage.
{ns03}:Element1;
```

You can declare a namespace, constant, and variable within a module or function. However, you can declare only a namespace or constant in schema scope (that is, outside a module scope).

For more information about using namespaces with DFDL, see **Namespaces** in the DFDL v1.0 Specification.

Querying null values in a message in the DFDL domain
You can use an ESQL statement to compare an element to NULL.

About this task

If you want to compare an element to NULL, code the statement:

```
IF InputRoot.DFDL.MyMessage.Elem2.Child1 IS NULL THEN
   DO:
    -- more ESQL --
END IF;
```

If nulls are permitted for this element, this statement tests whether the element exists in the input message, or whether it exists and contains one of the 'nil' values defined for the element in the DFDL schema. The returned result of this test is determined as follows:

- If the element Child1 is not in the bit stream, this test returns TRUE.
- If the element Child1 is in the bit stream and contains one of the 'nil' values, this test returns TRUE.
- If the element Child1 is in the bit stream and does not contain one of the 'nil' values, this test returns FALSE.

If you want to determine if the field is missing, rather than present but with null value, you can use the ESQL CARDINALITY function.

Setting null values in a message in the DFDL domain
You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

About this task

To set a value of an element in an output message, you normally code an ESQL statement, for example:

```
SET OutputRoot.DFDL.MyMessage.Elem2.Child1 = 'xyz';
```

or its equivalent statement:

```
SET OutputRoot.DFDL.MyMessage.Elem2.Child1 VALUE = 'xyz';
```

If you set the element to a non-null value, these two statements give identical results. However, if you want to set the value to null, these two statements do not give the same result:

Procedure

1. If you set the element to NULL using the following statement, the element is deleted from the message tree:

   ```
   SET OutputRoot.DFDL.MyMessage.Elem2.Child1 = NULL;
   ```

   This is called implicit null processing.
2. If you set the value of this element to NULL as follows:

   ```
   SET OutputRoot.DFDL.MyMessage.Elem2.Child1 VALUE = NULL;
   ```

   the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element.

   This is called explicit null processing.

Results

Setting a complex element to NULL deletes that element and all its children.

Working with DFDL messages and bit streams
Use the ASBITSTREAM function and the CREATE statement to manage DFDL-described message content.

About this task

Note: Note that FolderBitStream is not supported when manipulating messages in the DFDL domain.

The ASBITSTREAM function

If you code an ASBITSTREAM function to parse your message tree to a bit stream, and the `parser mode` option is set to RootBitStream , the result is a DFDL document that is built from the children of the target element in the normal way. This algorithm is identical to the algorithm that is used to generate the normal output bit stream. Because the target element is not included in the output bit stream, you must ensure that the children of the target element follow the constraints for a DFDL document.

One constraint is that there must be only one body element in the message. You can use a well-formed bit stream obtained in this way to re-create the original logical tree by using a CREATE statement that includes a PARSE clause.

For further information about the ASBITSTREAM function, and some examples of its use, see ASBITSTREAM function.

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause to parse your bit stream to a message tree, and the `parser mode` option is set to RootBitStream, the expected bit stream is a normal DFDL-described message. An element is created in the logical tree for each element that is identified in the bit stream. This algorithm is identical to the algorithm that is used when parsing a bit stream from an input node.

For further information about the CREATE statement, and examples of its use, see CREATE statement.

Manipulating messages in the XML domains
You can manipulate messages in the XML, XMLNS, and XMLNSC domains.

About this task

The following topics contain instructions about manipulating messages in the XMLNSC, XMLNS, and XML domains.

- “Working with XML messages” on page 1697
- “Manipulating messages in the XMLNSC domain” on page 1706
- “Manipulating messages in the XMLNS domain” on page 1719
- “Manipulating messages in the XML domain” on page 1727

For information about dealing with MRM XML messages, see “Manipulating messages in the MRM domain” on page 1727.

Working with XML messages
The following topics provide information about typical tasks for processing XML messages.

About this task

Some of this information is available publicly in Web pages and online tutorials. If you are new to XML, you will find it useful to also read about the XML standard.

- “Constructing an XML message tree” on page 1698
- “Working with namespaces” on page 1698
- “Working with binary data” on page 1699
- “XMLNSC: Working with CData” on page 1700
- “XMLNSC: Working with XML messages and bit streams” on page 1702

- “Working with large XML messages” on page 1703

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Constructing an XML message tree
When constructing an XML message tree, consider the order of fields in the tree.

About this task

Order of fields in the message tree

When you create an XML output message in a Compute node, the order of your lines of ESQL code is important, because the message elements are created in the order that you code them.

Consider the following XML message:

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
```

If you want to add a DocType Declaration to this, insert the DocType Declaration before you copy the input message to the output message.

For example:

```
SET OutputRoot.XMLNS.(XML.XmlDecl) = '';
SET OutputRoot.XMLNS.(XML.XmlDecl).(XML.Version) = '1.0';
SET OutputRoot.XMLNS.(XML.DocTypeDecl)Order ='';
SET OutputRoot.XMLNS.(XML.DocTypeDecl).(XML.SystemId)   = 'NewDtdName.dtd';
SET OutputRoot = InputRoot;
  -- more ESQL --
```

If you put the last statement to copy the input message before the XML-specific statements, the following XML is generated for the output message.

```
<Order>
  <ItemNo>1</ItemNo>
  <Quantity>2</Quantity>
</Order>
<?xml version="1.0"?>
```

This is not well-formed XML and causes an error when it is written from the message tree to a bit stream in the output node.

Setting the field type

If you copy a message tree from input to output without changing the domain, most of the syntax elements will be created by the parser (XMLNSC or XMLNS) and their field types will be correct. However, if you construct your message tree from a database query, or from another parser's message tree, you must ensure that you identify each syntax element correctly by using its field type. You can find full details of the field type constants used by XMLNSC and XMLNS in the following topics:

- “XMLNSC: Using field types” on page 533
- XML constructs

Working with namespaces
The following example shows how to use ESQL to work with namespaces.

About this task

Namespace constants are declared at the start of the main module so that you can use prefixes in the ESQL statements instead of the full URIs of the namespaces. The namespace constants affect only the ESQL; they do not control the prefixes that are generated in the output message. The prefixes in the generated output message are controlled by namespace declarations in the message tree. You can

include namespace declarations in the tree using the XML.NamespaceDecl field type. These elements are then used to generate namespace declarations in the output message.

When the output message is generated, if the parser encounters a namespace for which it has no corresponding namespace declaration, a prefix is automatically generated using prefixes of the form NSn where n is a positive integer.

```
CREATE COMPUTE MODULE xmlns_doc_flow_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
CALL CopyMessageHeaders();
-- Declaration of namespace constants --
These are only used by ESQL

DECLARE sp1 NAMESPACE 'http://www.ibm.com/space1';
DECLARE sp2 NAMESPACE 'http://www.ibm.com/space2';
DECLARE sp3 NAMESPACE 'http://www.ibm.com/space3';

-- Namespace declaration for prefix 'space1'
SET OutputRoot.XMLNS.message.(XML.NamespaceDecl)xmlns:space1 = 'http://www.ibm.com/space1';
SET OutputRoot.XMLNS.message.sp1:data1 = 'Hello!';

-- Default namespace declaration ( empty prefix )
SET OutputRoot.XMLNS.message.sp2:data2.(XML.NamespaceDecl)xmlns = 'http://www.ibm.com/space2';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData1 = 'Hola!';
SET OutputRoot.XMLNS.message.sp2:data2.sp2:subData2 = 'Guten Tag!';
SET OutputRoot.XMLNS.message.sp3:data3 = 'Bonjour!';
RETURN TRUE;
END;

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

When this ESQL is processed, the following output message is generated:

```
<message xmlns:space1="http://www.ibm.com/space1">
 <space1:data1>Hello!</space1:data1>
 <data2 xmlns="http://www.ibm.com/space2">
  <subData1>Hola!</subData1>
  <subData2>Guten Tag!</subData2>
 </data2>
 <NS1:data3 xmlns="http://www.ibm.com/space3>Bonjour!</NS1:data3>
</message>
```

Working with binary data
If you need to include binary data or non-valid characters in your XML documents, the safest method is to encode the data as a binary string.

About this task

Binary encodings for XML

There are two methods of encoding binary data in an XML document.

```
hexBinary: <nonXMLChars>000102030405060708B0C0E0F</nonXMLChars>

base64Binary: <nonXMLChars>AAECAwQFBgcICwwODw==</nonXMLChars>
```

The base64Binary encoding makes better use of the available XML characters, and on average a base64-encoded binary field is 2/3 the size of its hexBinary equivalent. Base64Binary is widely used by the MIME format and by various XML-based standards.

You might prefer to use the simpler hexBinary encoding if you are sending data to applications that cannot decode base64 data, and if the increase in message size is not important.

Parsing binary data

The most straightforward way to parse any binary data is to use the XMLNSC parser with a message model:

1. Locate or construct an XML Schema that describes your input XML.
2. In your message flow, set the node properties as follows:
   - On the Default page, set *Message Domain* to XMLNSC.
   - On the Validation page, set *Validation* to Content and Value.
   - In the XMLNSC properties, select the check box option *Build tree using XML Schema types*.

The XMLNSC parser automatically decodes your hexBinary or base64Binary data, being guided by the simple type of the element or attribute that contains the binary data. The message tree will contain the decoded BLOB value.

If you are using the XMLNS domain, you must parse the binary data as a string. It appears in the message tree as a CHARACTER value. If the data is encoded as hexBinary, you can use the ESQL CAST function to convert it to a BLOB value. If the data is encoded as base64Binary, the easiest approach is to use the function **BASE64DECODE**. For more information, see BASE64DECODE function.

Generating binary data

You can generate binary data in your output XML in either hexBinary or base64Binary encoding.

For hexBinary, use the ESQL CAST statement to convert your BLOB data to a hexBinary string.

For base64Binary, you have two options:

- Call the function **BASE64ENCODE**. For more information, see BASE64ENCODE function.
- Use the XMLNSC parser, and modify the type field on the syntax element, as shown in this example:

```
-- ESQL code to generate base64-encoded binary data
DECLARE myBLOB BLOB;
-- Populate myBLOB with your binary data
CREATE LASTCHILD OF OutputRoot.XMLNSC.message
       TYPE BITOR(XMLNSC.Attribute, XMLNSC.base64Binary)
       NAME myBase64Element
       VALUE myBLOB;
```

Note that setting the field type to XMLNSC.base64Binary does not change the logical value in the message tree. In your message flow, it is still a BLOB, and if you ask for its string representation, it is reported as a hexBinary string. However, when the message tree is converted to a bit stream (in an output node, or by a call to ASBITSTREAM) the base64 conversion is performed automatically, and the output XML contains the correct base64 string.

XMLNSC: Working with CData

A CData section can be used to embed an XML document within another XML document.

About this task

What is a CData section?

An XML element can contain text content:

```
<element>text content</element>
```

However, some characters cannot appear in that content. In particular, '<' and '&' both have special meaning to an XML parser. If they are included in the text content of an element, they change the meaning of the XML document.

For example, this is a badly formed XML document:

```
<element><text><content></element>
```

There are two ways to make the XML well-formed:

1. Use character entities:

```
<element>&lt;text&gt;&lt;content&gt;</element>
```

2. Use a CData section:

```
<element><![CDATA[<text><content>]]></element>
```

What can you use a CData section for?

In a CData section, you can include XML markup in the value of an element. However, non-valid XML characters cannot be included. Binary data also cannot be included in a CData section.

The most common use for CData is to embed one XML document within another. For example:

```
<outer>
    <embedXML>
        <![CDATA[<innerMsg></innerMsg>]]>
    </embedXML>
</outer>
```

You can even embed a badly-formed XML document in this way, because the XML parser does not attempt to parse the content of a CData section.

```
<outer>
    <embedXML>
        <![CDATA[<badXML></wrongClosingTag>]]>
    </embedXML>
</outer>
```

The following items are not valid within a CData section:

- Non-valid XML characters (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets)
- The text string ']]>' (because this terminates the CData section)

Because of these restrictions, do not use a CData section to include arbitrary text in your XML document, and do not try to use a CData section to hold binary data (unless it is encoded as hexBinary or base64Binary).

How do you add a CData section to an output XML message?

Consider the following input message :

```
 <TestCase>
    <Folder>
       <Field1>Value1</Field1>
       <Field2>Value2</Field2>
       <Field3>Value3</Field3>
    </Folder>
 </TestCase>
```

The following ESQL shows how to serialize a whole message:

```
 DECLARE wholeMsgBlob BLOB
   ASBITSTREAM(InputRoot.XMLNSC,
               InputRoot.Properties.Encoding,
               InputRoot.Properties.CodedCharSetId );
  DECLARE wholeMsgChar CHAR
   CAST(wholeMsgBlob AS CHAR CCSID InputRoot.Properties.CodedCharSetId);
 SET OutputRoot.XMLNSC.Output.(XMLNSC.CDataField)Field1 = wholeMsgChar;
```

This example serializes the InputRoot.XMLNSC.TestCase.Folder portion of the message tree.

If the output message tree were examined before an MQOutput node, this would show :

```
(0x01000010):XML        = (
  (0x01000000):Output = (
    (0x01000000):Field1 = (
      (0x02000001): = '<TestCase><Folder><Field1>Value1</Field1><Field2>Value2</Field2>
```

```
                        <Field3>Value3</Field3></Folder><Folder2><Field1>Value1</Field1>
                        <Field2>Value2</Field2><Field3>Value3</Field3><Folder2></TestCase>'
       )
       )
   )
```

As can be seen, each CData section contains a single scalar value that is the character representation of the portion of the XML message that is required.

This tree produces the following XML output message :

```
<Output>
 <Field1><![CDATA[<TestCase><Folder><Field1>Value 1</Field1>
                                    <Field2>Value 2</Field2>
                                    <Field3>Value 3</Field3></Folder>
                            <Folder2><Field1>Value 1</Field1>
                                     <Field2>Value 2</Field2>
                                     <Field3>Value 3</Field3></Folder2>
                  </TestCase>]]</Field1>
 </Output>
```

CData section handling

Use the XMLNSC parser to merge CData sections into a single element in the message tree. Run the following command, where *integrationNodeName* is the name of your integration node and *integrationServerName* is the name of your integration server:

```
mqsichangeproperties integrationNodeName -e integrationServerName  -o
 ComIbmGenericXmlParserFactory  -n cdataFieldMerging -v option
```

option can be one of the following values:

*Table 68. Options for the **-v** flag*

Value	Description
no	Split the CData section in the message tree based on the "]" character and amount of data. This is the default value.
yes	Merge CData section into a single element in the message tree

XMLNSC: Working with XML messages and bit streams
Use the ASBITSTREAM function and the CREATE statement to manage XML message content.

About this task

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to RootBitStream to parse a message tree to a bit stream, the result is an XML document that is built from the children of the target element in the normal way. This algorithm is identical to that used to generate the normal output bit stream. Because the target element is not included in the output bit stream, you must ensure that the children of the element follow the constraints for an XML document.

One constraint is that there must be only one body element in the message. You can use a well-formed bit stream obtained in this way to re-create the original tree using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to FolderBitStream to parse a message tree to a bit stream, the generated bit stream is an XML document built from the target element and its children. Any DocTypeDecl or XmlDecl elements are ignored, and the target element itself is included in the generated bit stream.

The advantage of this mode is that the target element becomes the body element of the document, and that body element can have multiple elements nested within it. Use this mode to obtain a bit stream description of arbitrary sub-trees owned by an XML parser. You can use bit streams obtained in

this way to re-create the original tree using a CREATE statement with a PARSE clause, and a mode of FolderBitStream.

For further information about the ASBITSTREAM function, and some examples of its use, see ASBITSTREAM function.

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause with the parser mode option set to RootBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. A field in the tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node. In particular, an element named 'XML', 'XMLNS', or 'XMLNSC' is created as the root element of the tree, and all the content in the message is created as children of that root.

If you code a CREATE statement with a PARSE clause with the parser mode option set to FolderBitStream to parse a bit stream to a message tree, the expected bit stream is a normal XML document. Any content outside the body element (such as an XML declaration or doctype) is discarded. The first element created during the parse corresponds to the body of the XML document, and from there the parse proceeds as normal.

For further information about the CREATE statement, and examples of its use, see CREATE statement.

Working with large XML messages
The tree representation of an XML message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code ESQL statements that help to reduce the storage load on the integration node.

About this task

When an input bit stream is parsed and a logical tree is created, the tree representation of an XML message is typically bigger, and in some cases much bigger, than the corresponding bit stream.

The reasons for this expansion include the following factors:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode, which can double the size
- The inclusion of field names that might have been implicit in the bit stream
- The presence of control data that is associated with operation of the integration node

Manipulating a large message tree can require much storage. If you design a message flow that handles large messages that are made up of repeating structures, you can code ESQL statements that help to reduce the storage load on the integration node. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the integration node to perform limited parsing of the message, and to keep in storage at one time, only that part of the message tree that reflects a single record. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree; for example, in the local environment.

This technique reduces the memory that is used by the integration node to the memory that is needed to hold the full input and output bit streams, plus the memory that is needed for the message trees of just one record. This technique also provides memory savings when even a few repeats are encountered in the message. The integration node uses partial parsing and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node, take any of the following steps.

- Copy the body of the input message as a bit stream to a special folder in the output message. This action creates a modifiable copy of the input message that is not parsed and therefore uses a minimum amount of memory.
- Avoid any inspection of the input message, which avoids the need to parse the message.

- Use a loop and a reference variable to step through the message one record at a time. For each record, use the following processes:
  - Use normal transforms to build a corresponding output subtree in a second special folder.
  - Use the ASBITSTREAM function to generate a bit stream for the output subtree. The generated bit stream is stored in a BitStream element that is placed in the position in the output subtree that corresponds to its required position in the final bit stream.
  - Use the DELETE statement to delete both the current input and output record message trees when you have completed their manipulation.
  - When you have completed the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages.

The following ESQL code provides an example of one implementation, and is a modification of the ESQL example in "Transforming a complex message" on page 1684. It uses a single SET statement with nested SELECT functions to transform a message that contains nested, repeating structures.

```
-- Copy the MQMD header
  SET OutputRoot.MQMD = InputRoot.MQMD;

-- Create a special folder in the output message to hold the input tree
-- Note : SourceMessageTree is the root element of an XML parser
  CREATE LASTCHILD OF OutputRoot.XMLNS.Data DOMAIN 'XMLNS' NAME 'SourceMessageTree';

-- Copy the input message to a special folder in the output message
-- Note : This is a root to root copy which will therefore not build trees
  SET OutputRoot.XMLNS.Data.SourceMessageTree = InputRoot.XMLNS;

-- Create a special folder in the output message to hold the output tree
  CREATE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Prepare to loop through the purchased items
  DECLARE sourceCursor REFERENCE TO OutputRoot.XMLNS.Data.SourceMessageTree.Invoice;
  DECLARE targetCursor REFERENCE TO OutputRoot.XMLNS.Data.TargetMessageTree;
  DECLARE resultCursor REFERENCE TO OutputRoot.XMLNS.Data;
  DECLARE grandTotal   FLOAT     0.0e0;

-- Create a block so that it's easy to abandon processing
  ProcessInvoice: BEGIN
  -- If there are no Invoices in the input message, there is nothing to do
    IF NOT LASTMOVE(sourceCursor) THEN
      LEAVE ProcessInvoice;
    END IF;

  -- Loop through the invoices in the source tree
  InvoiceLoop : LOOP
    -- Inspect the current invoice and create a matching Statement
    SET targetCursor.Statement = THE (SELECT 'Monthly' AS (XML.Attribute)Type,
                                          'Full' AS (0x03000000)Style[1],
                                         I.Customer.FirstName AS Customer.Name,
                                         I.Customer.LastName AS Customer.Surname,
                                            I.Customer.Title AS Customer.Title,
                                         (SELECT
                                           FIELDVALUE(II.Title) AS Title,
                                           CAST(II.UnitPrice AS FLOAT) * 1.6 AS Cost,
                                           II.Quantity AS Qty
                                         FROM I.Purchases.Item[] AS II
                                         WHERE II.UnitPrice> 0.0) AS Purchases.Article[],
                                         (SELECT
                                           SUM( CAST(II.UnitPrice AS FLOAT) *
                                                CAST(II.Quantity  AS FLOAT) *
                                                1.6                          )
                                         FROM I.Purchases.Item[] AS II) AS Amount,
                                         'Dollars' AS Amount.(XML.Attribute)Currency
                                       FROM sourceCursor AS I
                                       WHERE I.Customer.LastName <> 'White');

    -- Turn the current Statement into a bit stream
    DECLARE StatementBitStream BLOB
    ASBITSTREAM(targetCursor.Statement OPTIONS FolderBitStream);
    -- If the SELECT produced a result
    -- (that is, it was not filtered out by the WHERE clause),
    -- process the Statement
    IF StatementBitStream IS NOT NULL THEN
```

```
    -- create a field to hold the bit stream in the result tree
       CREATE LASTCHILD OF resultCursor
          Type  XML.BitStream
          NAME  'StatementBitStream'
          VALUE StatementBitStream;

    -- Add the current Statement's Amount to the grand total
    -- Note that the cast is necessary because of the behavior
    -- of the XML syntax element
       SET grandTotal = grandTotal
        + CAST(targetCursor.Statement.Amount AS FLOAT);
    END IF;

    -- Delete the real Statement tree leaving only the bit stream version
    DELETE FIELD targetCursor.Statement;

    -- Step onto the next Invoice,
    -- removing the previous invoice and any
    -- text elements that might have been
    -- interspersed with the Invoices

    REPEAT
      MOVE sourceCursor NEXTSIBLING;
      DELETE PREVIOUSSIBLING OF sourceCursor;
      UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
       OR (LASTMOVE(sourceCursor) = FALSE)
    END REPEAT;

    -- If there are no more invoices to process, abandon the loop
    IF NOT LASTMOVE(sourceCursor) THEN
     LEAVE InvoiceLoop;
    END IF;

  END LOOP InvoiceLoop;
END ProcessInvoice;

-- Remove the temporary source and target folders
DELETE FIELD OutputRoot.XMLNS.Data.SourceMessageTree;
DELETE FIELD OutputRoot.XMLNS.Data.TargetMessageTree;

-- Finally add the grand total
SET resultCursor.GrandTotal = grandTotal;
```

This ESQL code produces the following output message:

```
<Data>
 <Statement Type="Monthly" Style="Full">
  <Customer>
   <Name>Andrew</Name>
   <Surname>Smith</Surname>
   <Title>Mr</Title>
  </Customer>
  <Purchases>
   <Article>
    <Title>The XML Companion</Title>
    <Cost>4.472E+1</Cost>
    <Qty>2</Qty>
   </Article>
   <Article>
    <Title>A Complete Guide to DB2 Universal Database</Title>
    <Cost>6.872E+1</Cost>
    <Qty>1</Qty>
   </Article>
   <Article>
    <Title>JAVA 2 Developers Handbook</Title>
    <Cost>9.5984E+1</Cost>
    <Qty>1</Qty>
   </Article>
  </Purchases>
  <Amount Currency="Dollars">2.54144E+2</Amount>
  </Statement>
  <GrandTotal>2.54144E+2</GrandTotal>
 </Data>
```

Manipulating messages in the XMLNSC domain
If you are writing ESQL to process messages in the XMLNSC domain, it is helpful to learn about the structure of the message tree that the XMLNSC parser builds.

About this task

The topics in this section provide information about how to write ESQL for processing messages that belong to the XMLNSC domain, and that are parsed by the XMLNSC parser. For background information, see "XMLNSC parser" on page 531.

The following topics provide detailed information about the structure of the message tree that the XMLNSC parser builds, and the field types that it uses.

- "XMLNSC: The XML declaration" on page 1706
- "XMLNSC: The inline DTD" on page 1707
- "XMLNSC: The message body" on page 1707
- "XMLNSC: XML Schema support" on page 1715

If you are migrating from XML, XMLNS, or MRM XML, see "Migrating to XMLNSC" on page 1718.

For further information about processing XML messages, see "Working with XML messages" on page 1697.

XMLNSC: The XML declaration
The XML declaration is represented in the message tree by a syntax element with field type XMLNSC.XMLDeclaration.

If an XML declaration is created by the XMLNSC parser, its name is 'XmlDeclaration'. However, when a message tree is being produced, the name is not important: the XMLNSC parser recognizes this syntax element by its field type only. The following example shows a typical declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration has three optional attributes; Version, Standalone, and Encoding. The XMLNSC parser does not define special field types for these attributes. Instead, they are identified by their name, and by their position as a child of the XML Declaration element.

ESQL example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, code the following ESQL statements:

```
CREATE FIRSTCHILD OF OutputRoot.XMLNSC TYPE XMLNSC.XmlDeclaration NAME 'XmlDeclaration';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Version = '1.0';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Encoding = 'UTF-8';
SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)StandAlone = 'yes';
```

The first line is optional; if it is omitted, the XMLNSC.XMLDeclaration element is automatically created when it is referenced by the second line.

Java example code to create an XML declaration

To construct the XML declaration that is shown in the previous example, write the following Java code:

```
//Create the XML domain root node
MBElement xmlRoot =
 root.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
//Create the XML declaration parent node
MbElement xmlDecl =
 xmlRoot.createElementAsFirstChild(MbXMLNSC.XML_DECLARATION);

xmlDecl.setName("XmlDeclaration");

MbElement version =
```

```
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Version", "1.0");
MbElement encoding =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Encoding", "utf-8");
MbElement standalone =
 xmlDecl.CreateElementAsFirstChild(MbXMLNSC.ATTRIBUTE, "Standalone", "yes");
```

Note: In both the ESQL example and the Java example, 'Version', 'StandAlone', and 'Encoding' can all be written in lowercase.

XMLNSC: The inline DTD
When you parse an XML document that has an inline DTD, the XMLNSC parser does not put the DTD information into the message tree. However, by using ESQL code, you can add XML entity definitions to the message tree, and these definitions are used when the message tree is produced by the XMLNSC parser.

ESQL example code for entity definition and entity reference

This example assumes that InputRoot.XMLNSC has been created from the following XML message:

```
<BookInfo dtn="BookInfo" edn="author" edv="A.N.Other"/>
```

The following output message is generated:

```
<!DOCTYPE BookInfo [<!ENTITY author "A.N.Other">]>
<BookInfo><entref>&author;</entref></BookInfo>
```

The ESQL to create the output message is:

```
DECLARE cursor REFERENCE TO InputRoot.XMLNSC.BookInfo;
DECLARE docTypeName CHARACTER cursor.dtn;
DECLARE authorRef CHARACTER 'author';
-- Create <!DOCTYPE BOOKInfo …
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType)* NAME = docTypeName;
-- Create  <!ENTITY author "A.N.Other" > …
SET OutputRoot.XMLNSC.(XMLNSC.DocumentType){docTypeName}.(XMLNSC.EntityDefinition) {authorRef} =
cursor.edv;
-- Create the entity reference
SET OutputRoot.XMLNSC.(XMLNSC.Folder){docTypeName}.(XMLNSC.EntityReference)entref = authorRef;
```

XMLNSC: The message body
The XMLNSC parser builds a message tree from the body of an XML document.

The following topics describe how the XMLNSC parser builds a message tree from the body of an XML document:

- “XMLNSC: Using field types” on page 533
- “XMLNSC: Attributes and elements” on page 1710
- “XMLNSC: Namespace declarations” on page 1711
- “XMLNSC: Element values and mixed content” on page 1712
- “XMLNSC: Comments and Processing Instructions” on page 1715

XMLNSC: Using field types
The XMLNSC parser sets the field type on every syntax element that it creates.

The field type indicates the type of XML construct that the element represents. The XMLNSC parser uses the field type when writing a message tree. The field type can be set by using ESQL or Java to control the output XML. The field types that are used by the XMLNSC parser must be referenced by using constants with names that are prefixed by 'XMLNSC.'

Tip: Field type constants that have the prefix 'XML.' are for use with the XMLNS and XML parsers only, and are not valid with the XMLNSC or MRM parsers.

Field types for creating syntax elements

Use the following field type constants to create syntax elements in the message tree. The XMLNSC parser uses these values when creating a message tree from an input message.

XML construct	XMLNSC Field Type constant	Value
Simple Element	XMLNSC.Field XMLNSC.CDataField	0x03000000 0x03000001
Attribute	XMLNSC.SingleAttribute XMLNSC.Attribute	0x03000101 0x03000100
Mixed content	XMLNSC.Value XMLNSC.CDataValue	0x02000000 0x02000001
Namespace Declaration	XMLNSC.SingleNamespaceDecl XMLNSC.NamespaceDecl	0x03000102 0x03000103
Complex element	XMLNSC.Folder	0x01000000
Inline DTD	XMLNSC.DocumentType	0x01000300
XML declaration	XMLNSC.XmlDeclaration	0x01000400
Entity reference	XMLNSC.EntityReference	0x02000100
Entity definition	XMLNSC.SingleEntityDefinition XMLNSC.EntityDefinition	0x03000301 0x03000300
Comment	XMLNSC.Comment	0x03000400
Processing Instruction	XMLNSC.ProcessingInstruction	0x03000401

Field types for path expressions (generic field types)

Use the following field type constants when querying the message tree by using a path expression; for example:

```
SET str = FIELDVALUE(InputRoot.e1.(XMLNSC.Attribute)attr1)
```

It is good practice to specify field types when querying a message tree built by the XMLNSC parser. This makes your ESQL code more specific and more readable, and it avoids incorrect results in some cases. However, care is required when choosing which field type constant to use. When you use the XMLNSC parser, use a generic field type constant when querying the message tree. This allows your path expression to tolerate variations in the input XML.

The generic field type constants are listed in the following table:

XML construct	XMLNSC Field Type constant	Purpose
Tag	XMLNSC.Element	Matches any tag, whether it contains child tags (XMLNSC.Folder) or a value (XMLNSC.Field)

XML construct	XMLNSC Field Type constant	Purpose
Element	XMLNSC.Field	Matches a tag which contains normal text, CData, or a mixture of both. Does not match tags which contain child tags.
Attribute	XMLNSC.Attribute	Matches single-quoted and double-quoted attributes
Mixed content	XMLNSC.Value	Matches normal text, CData, or a mixture of both
XML Declaration	XMLNSC.NamespaceDecl	Matches single- and double-quoted declarations

If you write

```
InputRoot.e1.(XMLNSC.DoubleAttribute)attrName
```

your path expression does not match a single-quoted attribute. If you use the generic field type constant *XMLNSC.Attribute*, your message flow works with either single-quoted or double-quoted attributes.

Note that you should always use the field type constants and not their numeric values.

Field types for controlling output format

The following field types are provided for XML Schema and base64 support. Do not use these field type constants in path expressions; use them in conjunction with XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE and BLOB values. See "XMLNSC: XML Schema support" on page 1715 for further information.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	The value must be a DATE. If the field type includes this value, the DATE value is produced by using the XML Schema gDay format.	0x00000030
XMLNSC.base64Binary	The value must be a BLOB. The value is produced with base64 encoding.	0x00000060
XMLNSC.List	The element must be XMLNSC.Attribute or XMLNSC.Field. If the field type includes this value, the values of all child elements in the message tree are produced as a space-separated list.	0x00000070

Field types for direct output

Use the following field types to produce pre-constructed segments of an XML document. Character escaping is not done; therefore, take extra care not to construct a badly-formed output document. Use these constants only after carefully exploring alternative solutions.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.BitStream	The value of this syntax element must be a BLOB. The value is written directly to the output bit stream. For more information about its usage, see "Working with large XML messages" on page 1703.	0x03000200
XMLNSC.AsisElementContent	The value of this syntax element must be CHARACTER. The value is written directly to the output bit stream. No character substitutions are performed. Use this element with care.	0x03000600

XMLNSC: Attributes and elements

The XMLNSC parser uses field types to represent attributes and elements.

Use the following field type constants when creating your own syntax elements in the message tree.

Table 69. Specific field type constants

XML Construct	XMLNSC Field Type constant	Value
Complex element	XMLNSC.Folder	0x01000000
Simple element	`XMLNSC.Field` `XMLNSC.CDataField`	`0x02000000` `0x02000001`
Attribute	`XMLNSC.SingleAttribute` `XMLNSC.Attribute`	`0x03000100` `0x03000101`

When accessing elements and attributes in the message tree, use generic field type constants which match all of the alternative values. Because there is only one type of Folder element, it is safe to use XMLNSC.Folder when querying the message tree.

Table 70. Generic field type constants

XML Construct	XMLNSC Field Type constant	Purpose
Element	XMLNSC.Field	Matches elements that contain normal text, CData, or a mixture of both
Attribute	XMLNSC.Attribute	Matches both single-quoted and double-quoted attributes

ESQL code examples

The following examples use this XML message:

```
<root id="12345">
    <id>ABCDE</id>
</root>
```

Note that the message contains an attribute and an element with the same name.

Example 1 : Query the value of an XML element

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Field)id)
```

The result is that value is set to 'ABCDE'.

Example 2 : Query the value of an XML attribute

```
SET value = FIELDVALUE(InputRoot.XMLNSC.root.(XMLNSC.Attribute)id)
```

The result is that value is set to '12345'.

Example 3 : Create the example message by using ESQL

```
CREATE LASTCHILD OF OutputRoot.XMLNSC Type XMLNSC.Folder Name 'root';
-- Note : XMLNSC.Attribute could be used here as well.
SET OutputRoot.XMLNSC.root.(XMLNSC.Attribute)id = '12345';
SET OutputRoot.XMLNSC.root.(XMLNSC.Field)id = 'ABCDE';
```

The first line is optional because the element 'root' is created automatically by the following line if it does not already exist.

XMLNSC: Namespace declarations
The XMLNSC parser provides full support for namespaces.

The XMLNSC parser sets the correct namespace on every syntax element that it creates while parsing a message, and stores namespace declarations in the message tree. The parser uses the namespace declarations to select the appropriate prefixes when outputting the message tree.

The XMLNSC parser uses the following field types to represent namespace declarations. Use the field type constants that are listed in this table when you create namespace declarations in the message tree.

Table 71. Specific field type constants

XML construct	XMLNSC field type constant	Value
Namespace declaration	`XMLNSC.SingleNamespaceDecl` `XMLNSC.DoubleNamespaceDecl`	`0x03000102` `0x03000103`

When accessing elements and attributes in the message tree, do not use the constants that are listed in the previous table; instead, use the generic field type constant that matches both of the values in the table above.

Table 72. Generic field type constants

XML construct	XMLNSC field type constant	Purpose
Namespace declaration	XMLNSC.NamespaceDecl	Matches namespace declarations in both single and double quotation marks

ESQL code examples

Example 1: Declaring a non-empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns:ns1 = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<ns1:root xmlns:ns1="namespace1">
    <ns1:example>ABCDE</ns1:example>
</ns1:root>
```

Note that the NAMESPACE constant space1 is just a local variable in the ESQL; it does not affect the namespace prefix ns1 that is defined by the NameSpaceDecl element and appears in the output message.

However, as shown here, space1 can be used to initialize the NameSpaceDecl for ns1. This avoids the need to duplicate the namespace URI ('namespace1' in this example), which in practice is typically a much longer string.

Example 2: Declaring an empty prefix

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.space1:root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.space1:root.space1:example = 'ABCDE';
```

This creates the following XML message:

```
<root xmlns="namespace1">
    <example>ABCDE</example>
</root>
```

Note that the syntax elements root and example must have a non-empty namespace. The default namespace declaration means that any child element without a prefix is in the namespace namespace1.

Example 3: Example of incorrect usage

```
DECLARE space1 NAMESPACE 'namespace1';
SET OutputRoot.XMLNSC.root.(XMLNSC.NamespaceDecl)xmlns = space1;
SET OutputRoot.XMLNSC.root.example = 'ABCDE';
```

This example causes the XMLNSC parser to issue the message BIP5014 when it attempts to create the message tree. The elements root and example are both within the scope of the default namespace declaration; therefore, in ESQL, these elements must be qualified by a namespace prefix bound to that namespace.

Example 4: Adding a namespace declaration with a prefix

```
SET OutputRoot.(XMLNSC.DoubleNamespaceDecl)xmlns:ns2 = space1;
```

This example of a SET statement creates a namespace declaration with the name ns2 in the namespace xmlns.

```
CREATE LASTCHILD OF OutputRoot IDENTITY (XMLNSC.DoubleNamespaceDecl)xmlns:ns2 VALUE space1;
```

```
CREATE LASTCHILD OF OutputRoot TYPE XMLNSC.DoubleNamespaceDecl NAMESPACE 'xmlns' NAME 'ns2'
 VALUE space1;
```

These examples of a CREATE statement also create a namespace declaration with the name ns2 in the namespace xmlns.

However, be aware that the following example of a CREATE statement creates a namespace declaration with the name xmlns:ns2 in the default namespace:

```
CREATE LASTCHILD OF OutputRoot TYPE XMLNSC.DoubleNamespaceDecl NAME 'xmlns:ns2' VALUE space1;
```

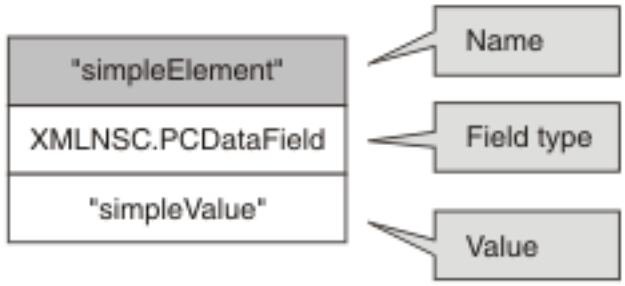
XMLNSC: Element values and mixed content

The XMLNSC parser is a compact parser; therefore, an element with single content is parsed as a single syntax element. When an element has both child elements and some text, the text is called *mixed content*.

Element with simple content

The following XML fragment with a single content is parsed as a single syntax element:

```
<simpleElement>simpleValue</simpleElement>
```



The value of this element can be queried with this ESQL:

```
SET val = FIELDVALUE(InputRoot.XMLNSC.(XMLNSC.Field)simpleElement);
```

To generate an element with simple content in the output:

```
SET OutputRoot.XMLNSC.(PCDataField)simpleElement VALUE = 'simpleValue';
```

Note that XMLNSC.Field is used when querying the message tree, but XMLNSC.PCDataField is specified when constructing the output message. XMLNSC.PCDataField can be used to query the message tree; however, that would not work if the input message used a CData section, as shown in the following example:

```
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

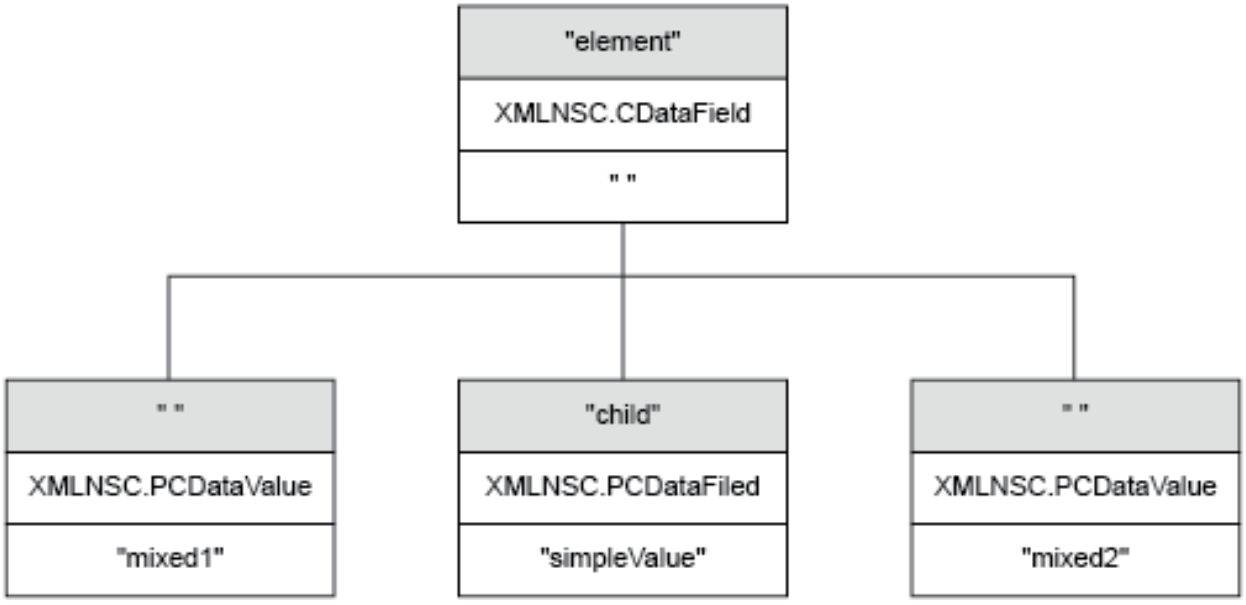
Element with mixed content

If an element has child elements, it is typically a 'folder', and does not have a value. When an element has both child elements and some text, the text is called 'mixed content'.

```
<element>mixed1<child>simpleValue</child>mixed2</element>
```

By default, mixed content is discarded because it is typically just formatting white space and has no business meaning. Mixed content can be preserved if you select the `Retain mixed content` check box on the Parser Options page of the node properties.

If mixed content is being preserved, the XMLNSC parser creates a Value child element for each distinct item of mixed content.



The mixed content can be queried with this ESQL:

```
SET mixed1 = FIELDVALUE(InputRoot.XMLNSC.(element).*[1];
```

The ESQL to construct the above XML fragment is:

```
CREATE ref REFERENCE TO OutputRoot.XMLNSC.element;
CREATE FIRSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed1';
CREATE LASTCHILD OF ref NAME 'child' TYPE XMLNSC.PCDataField  VALUE 'simpleValue';
CREATE LASTSTCHILD OF ref TYPE XMLNSC.PCDataValue VALUE 'mixed2';
```

The following ESQL enables the *Retain mixed content* option:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot
    PARSE(X OPTIONS XMLNSC.MixedContentRetainAll);
```

Element containing a CData section

A CData section is an XML notation that allows XML markup characters to be included in the content of an element.

The following two XML fragments are identical in their meaning:

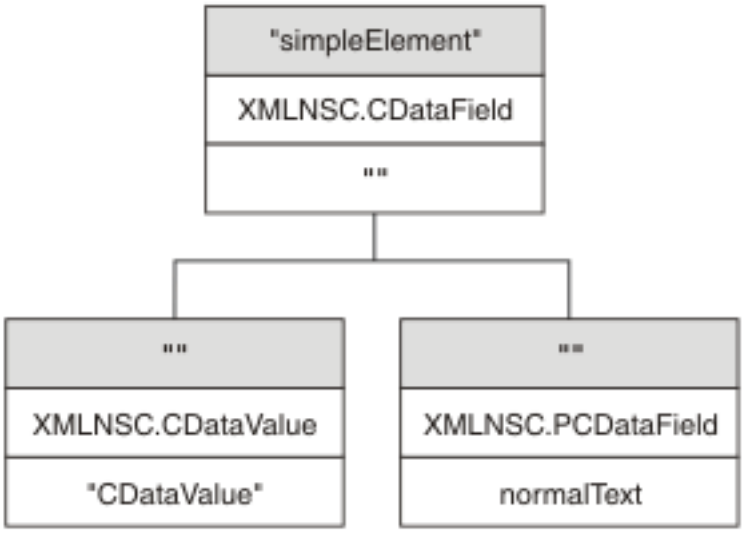
```
<simpleElement>simpleValue</simpleElement>
```

```
<simpleElement><![CDATA[simpleValue]]></simpleElement>
```

If the CData section is the only text content, the XMLNSC parser remembers that the input document contained a CData section by setting the field type to XMLNSC.CDataField instead of XMLNSC.PCDataField.

If the CData section is not the only text content, it is created as a child value element, with other child value elements representing the remaining text content. The following is an example of this:

```
<simpleElement><![CDATA[CDataValue]]>normalText</simpleElement>
```



See “XMLNSC: Working with CData” on page 1700 for more information about the correct use of CData in XML documents.

XMLNSC: Comments and Processing Instructions
The XMLNSC parser discards comments and processing instructions because both comments and processing instructions are auxiliary information with no business meaning.

Comments

Comments can be preserved if you select the *Retain comments* check box on the Parser Options page of the node properties.

If *Retain comments* is selected, each comment in the input document is represented by a single syntax element with field type XMLNSC.Comment. The *Retain comments* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME preserveComments
          OPTIONS XMLNSC.CommentsRetainAll);

-- do it again, this time discarding comments
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardComments
          OPTIONS XMLNSC.CommentsRetainNone);
```

Processing Instructions

Processing instructions can be preserved if you select the *Retain processing instructions* check box on the Parser Options page of the node properties.

If *Retain processing instructions* is selected, each processing instruction the input document is represented by a single syntax element with field type XMLNSC.ProcessingInstruction. The *Retain processing instructions* option can also be accessed by using the following ESQL:

```
DECLARE X BLOB;
-- assume that X contains an XML document
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME preserveProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainAll);

-- do it again, this time discarding processing instructions
CREATE LASTCHILD OF OutputRoot.XMLNSC
    PARSE(X DOMAIN XMLNSC
          NAME discardProcessingInstructions
          OPTIONS XMLNSC.ProcessingInstructionsRetainNone);
```

XMLNSC: XML Schema support
Use the XMLNSC parser to parse and validate by using an XML Schema.

For information about how to configure the XMLNSC parser to use an XML Schema, see "XMLNSC parser" on page 531.

The following topics describe how the XMLNSC parser uses the field type to hold information about XML Schema simple types. This behavior enables the parser to write dates and binary elements in the same form in which they were parsed, and according to the XML Schema specification. It also allows the message flow developer to write dates, lists, and binary data in the correct XML Schema format.

- "XMLNSC: base64 support" on page 1716
- "XMLNSC: XML Schema date formatting" on page 1716
- "XMLNSC: XML List type support" on page 1717

XMLNSC: base64 support
The XMLNSC parser can produce binary data in base64-encoded format.

If `Validation` is set to `Content and Value`, and `Build tree using schema types` is enabled, the XMLSNC parser automatically decodes base64 data and creates a BLOB value in the message tree. When producing a message tree, the XMLNSC parser will 'base64-encode' a BLOB if the field type includes the constant *XMLNSC.base64Binary*.

ESQL code example to output base64 data

```
DECLARE Base64Data BLOB '0102030405060708090A0B0C0D0E0F';
-- Add in the base64Binary field type
DECLARE base64FieldType INTEGER XMLNSC.Field + XMLNSC.base64Binary;
CREATE LASTCHILD OF OutputRoot DOMAIN 'XMLNSC' NAME 'XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE base64FieldType NAME 'myBinaryData' VALUE Base64Data;
```

Result : <myBinaryData>AQIDBAUGBwgJCgsMDQ4P</myBinaryData>

Note that this example does not depend on validation. The XMLNSC parser can produce base64 binary data even if `Validation` is set to `None`.

XMLNSC: XML Schema date formatting
The XMLNSC parser can parse and write all of the XML Schema simple types.

The rarely used types gYear, gYearMonth, gMonth, gMonthDay, and gDay do not map directly to an integration node data type. For these simple types, the XMLNSC parser adds one of the following constant values to the field type. This behavior allows the parser to produce the data for output in the same format as it was received.

Field types for controlling date format

The following field types are provided for XML Schema date format support. Do not use these field type constants in path expressions. Use them in conjunction with constants XMLNSC.Attribute and XMLNSC.Field to indicate the required output format for DATE values.

XMLNSC Field Type constant	Purpose	Value
XMLNSC.gYear	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYear format.	0x00000010
XMLNSC.gYearMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gYearMonth format.	0x00000040
XMLNSC.gMonth	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonth format.	0x00000020
XMLNSC.gMonthDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gMonthDay format.	0x00000050
XMLNSC.gDay	Value must be a DATE. If the field type includes this value, the DATE value is written by using the XML Schema gDay format.	0x00000030

ESQL code example

```
DECLARE gYear DATE ‘2007-01-01';
-- Add in the gYear field type
DECLARE gYearFieldType INTEGER XMLNSC.Field + XMLNSC.gYear;
CREATE LASTCHILD OF OutputRoot DOMAIN ‘XMLNSC' NAME ‘XMLNSC';
CREATE LASTCHILD OF OutputRoot.XMLNSC TYPE gYearFieldType NAME ‘gYear' VALUE gYear;
```

Result : <gYear>2007</gYear>

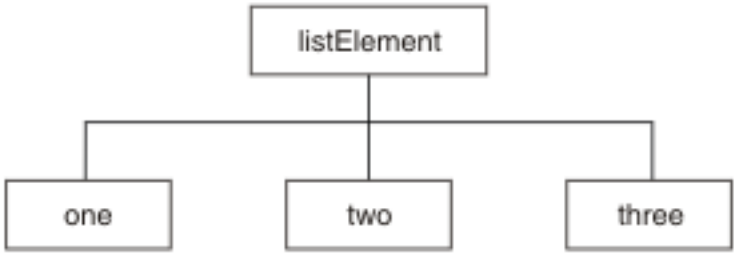
XMLNSC: XML List type support
The XMLNSC parser can automatically parse a space-separated list of values into individual syntax elements in the message tree if you select certain options.

An element or an attribute can have multiple values separated by spaces, as shown in the following examples:

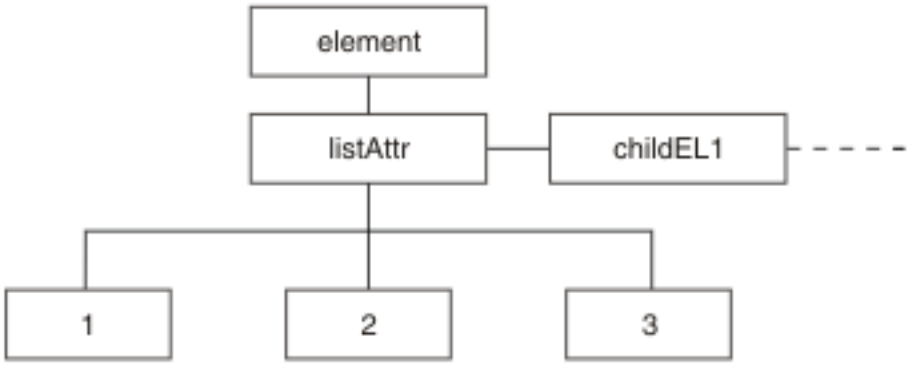
```
<listElement>one two three</listElement>
```

```
<element listAttribute="1 2 3"><childEL1/></element>
```

If your XML schema specifies a list type for an element or an attribute, and `Validation` is set to `Content and Value`, and `Build tree using schema types` is enabled, the XMLNSC parser automatically parses the space-separated list of values into individual syntax elements in the message tree. The resulting message tree looks like this:



and for an attribute with a list value it looks like this:



ESQL code examples

Access the individual values in a list

```
SET val = InputRoot.XMLNSC.listElement.*[1];
```

Result : val = 'one'

```
SET val = InputRoot.XMLNSC.element.(XMLNSC.Attribute)listAttr.*[3];
```

Result : val='3'

Create a list element in the message tree

```
CREATE LASTCHILD OF OutputRoot.XMLNSC
   Name 'listElement'
   Type XMLNSC.List;
DECLARE listEl REFERENCE TO OutputRoot.XMLNSC.listElement;
DECLARE listValType INTEGER XMLNSC.PCDataValue;
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'one';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'two';
CREATE LASTCHILD OF listEl TYPE listValType VALUE 'three';
```

Migrating to XMLNSC
The XMLNSC parser offers the best combination of features and performance for most applications.

Reasons to migrate

If your message flow uses the XMLNS or XML domain, you might want to migrate a message flow to XMLNSC to take advantage of the XML schema validation. If your message flow uses the MRM domain, you might want to migrate to XMLNSC to obtain standards-compliant validation, and a large reduction in processor usage.

Migrating from the XMLNS or XML domain

The XMLNSC parser differs from the XMLNS parser in the following ways:

- The XMLNSC parser builds a compact message tree.
- It uses different field type constants.
- It discards inline DTDs

In most cases, the compact message tree has no effect on ESQL paths or XPath expressions. Typically, a simple message tree query produces the same results in XMLNSC as in the XMLNS or XML domain. Changing the correlation name from XMLNS to XMLNSC is often sufficient, but care must be taken with the following items:

- Empty elements and null values.

  The XMLNSC parser does not always handle empty elements and null values in the same way as XML and XMLNS.
- Complex XPath expressions that navigate to the value of an element, then to its parent in a single query.

  These expressions might produce different results in the XMLNSC domain.

The field type constants that are used by the XMLNSC parser are different from those constants used by XMLNS or XML. Every occurrence of XML.Attribute, XML.XmlDecl, for example, must be changed to use the equivalent XMLNSC field type constant.

The discarding of inline DTDs only affects message flows that process the DTD.

Migrating from MRM XML

The XMLNSC parser differs from the MRM XML parser in the following ways:

- The XMLNSC parser uses field types to identify the XML constructs in the message tree. The MRM parser distinguishes attributes from elements by matching the message tree against the message definition.
- When writing a message tree, the XMLNSC parser selects namespace prefixes by detecting and using xmlns attributes in the message tree. The MRM XML parser uses a table in the message set properties.
- The MRM parser does not include the root tag of the XML document in the message tree.

Migrating a message flow from MRM to XMLNSC typically requires extensive changes to your message flow. However, the migration usually delivers a large reduction in processor usage, and allows much more accurate control of the output XML.

Manipulating messages in the XMLNS domain
When you write ESQL for processing messages in the XMLNS domain, it is helpful to understand the structure of the message tree that the XMLNS parser builds.

About this task

The topics in this section provide information about how to write ESQL for processing messages that belong to the XMLNS domain, and that are parsed by the XMLNS parser. Most of the information in these topics also applies to the XML domain, unless it refers to features that are not supported in the XML domain.

Refer to "XMLNS parser" on page 541 for background information.

The following topics provided detailed information about the structure of the message tree that the XMLNS parser builds, and the field types that it uses.

- "XMLNS: The XML declaration" on page 1719
- "XMLNS: The DTD" on page 1720
- "XMLNS: The XML message body" on page 1721

You can find more information about processing XML messages in "Working with XML messages" on page 1697.

XMLNS: The XML declaration
The beginning of an XML message can contain an XML declaration.

The following is an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

The XML Declaration is represented by the following types of syntax element:

- "XML.XMLDecl" on page 1719
- "XML.version" on page 1719
- "XML.standalone" on page 1720

"XMLNS: XML declaration example" on page 1720 includes another example of an XML declaration and the tree structure that it forms.

XML.XMLDecl
The XML Declaration is represented in the message tree by a syntax element with field type 'XML.XMLDecl'.

If the XML declaration is created by the XMLNS parser its name is 'XMLDecl'. However, when a message tree is being written, the name is not important; only the field type is used by the parser.

The following shows an example of a declaration:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

XML.version
The XML version attribute in the XML declaration is represented in the message tree by a syntax element with field type 'XML.version'.

The value of the XML version attribute is the value of the version attribute. It is always a child of an XML.XmlDecl syntaxelement. In the following example, the version element contains the string value 1.0:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

XML.encoding
The encoding attribute is represented by a syntax element with field type ‘XML.encoding', and is always a child of an XML.XmlDecl syntax element.

In the following example, the encoding attribute has a value of UTF-8.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

You cannot specify IBM MQ encodings in this element.

In your ESQL, (XML,"Encoding") must include quotation marks, because Encoding is a reserved word.

XML.standalone
The XML standalone element defines the existence of an externally-defined DTD. In the message tree it is represented by a syntax element with field type XML.standalone.

The value of the XML standalone element is the value of the standalone attribute in the XML declaration. It is always a child of an XML.XmlDecl syntax element. The only valid values for the standalone element are yes and no. The following is an example of this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE s1 PUBLIC "http://www.ibm.com/example.dtd" "example.dtd">
<s1>........</s1>
```

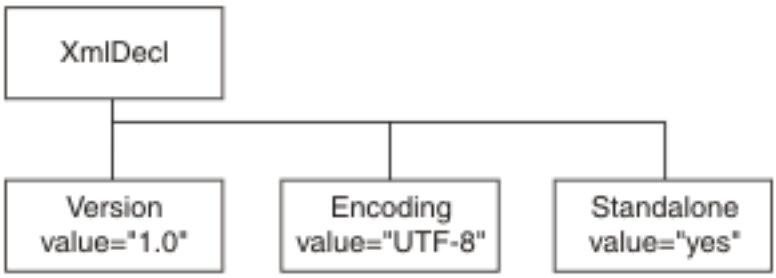
A value of no indicates that this XML document is not standalone, and depends on an externally-defined DTD. A value of yes indicates that the XML document is self-contained. However, because the current release of IBM App Connect Enterprise does not resolve externally-defined DTDs, the setting of standalone is irrelevant, and is ignored.

XMLNS: XML declaration example
The XMLNS parser creates a tree that represents an XML declaration.

The following example shows an XML declaration in an XML document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

The following figure shows the tree structure that the XMLNS parser creates for this declaration:



XMLNS: The DTD
The document type declaration (DTD) of an XML message is represented by a syntax element with field type XML.DocTypeDecl, and its children. These comprise the DOCTYPE construct.

Only internal (inline) DTD subsets are represented in the syntax element tree. An inline DTD is a DTD that is declared within the XML document itself. It can be a complete DTD definition, or it can extend the definition in an external DTD.

External DTD subsets (identified by the SystemID or PublicId elements described later in this section) can be referenced in the message, but those referenced are not resolved by the integration node.

The following field type constants can be used to reference the various parts of a DTD in the message tree:

- XML DocTypeDecl

- XML NotationDecl
- XML entities
- XML ElementDef
- XML AttributeList
- XML AttributeDef
- XML DocTypePI
- XML WhiteSpace and DocTypeWhiteSpace
- XML DocTypeComment

XML DTD example shows an example of an XML DTD.

See XML document type declaration for more information about handling an inline DTD.

XMLNS: The XML message body
Every XML message must have a body. The body consists of a hierarchy of XML elements and other XML constructs that represent the message data.

The XMLNS parser assigns a field type to every syntax element that it creates. The value of the field type indicates the XML construct that it represents. In the following topics, each field type is discussed, with a series of example XML fragments.

The following common element types are discussed:

- “XML.element” on page 1723
- “XML.Attribute” on page 1723

“XML message body example” on page 1727 provides an example of an XML message body and the tree structure that is created from it using the syntax elements types that are listed above.

More complex XML messages might use some of the following syntax element types:

- “XML.CDataSection” on page 1724
- “XML.EntityReferenceStart and XML.EntityReferenceEnd” on page 1725
- “XML.comment” on page 1725
- “XML.ProcessingInstruction” on page 1725
- “XML.AsisElementContent” on page 1726
- “XML.BitStream” on page 1726

Accessing attributes and elements
The XMLNS parser sets the field type on every message tree element that it creates.

The field type indicates the type of XML construct that the element represents. The field types used by the XMLNS parser can be referenced using constants with names prefixed with ‘XML.' Field type constants with prefix ‘XML.' are for use with the XMLNS and XML parsers only; they do not work with the XMLNSC or MRM parsers.

	XMLNS Field Type constant
Tag	XML.Element
Attribute	`XML.Attribute` `XML.Attr`

By using the field type in this way, the XMLNS parser can distinguish between an element and an attribute that have the same name.

Example XML

```
<parent id="12345">
```

```
    <id>ABCDE</id>
</parent>
```

Example ESQL

```
SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Element)id)
Result : value is 'ABCDE'

SET value = FIELDVALUE(InputRoot.XMLNS.parent.(XML.Attr)id)
Result : value is '12345'
```

Example using SELECT to access multiple attributes

In the Example message, the element Title within each Item element has three attributes: Category, Form, and Edition. For example, the first Title element contains:

```
<Title Category="Computer" Form="Paperback" Edition="2">The XML Companion</Title>
```

The element `InputRoot.XML.Invoice.Purchases.Item[1].Title` has four children in the logical tree: Category, Form, Edition, and the element value, which is "The XML Companion".

If you want to access the attributes for this element, you can code the following ESQL. This extract of code retrieves the attributes from the input message and creates them as elements in the output message. It does not process the value of the element itself in this example.

```
-- Set the cursor to the first XML.Attribute of the Title.
-- Note the * after (XML.Attribute) meaning any name, because the name might not be known
DECLARE cursor REFERENCE TO InputRoot.XMLNS.Invoice.Purchases.Item[1].Title.(XML.Attribute)*;
WHILE LASTMOVE(cursor) DO
-- Create a field with the same name as the XML.Attribute
-- and set its value to the value of the XML.Attribute

  SET OutputRoot.XML.Data.Attributes.{FIELDNAME(cursor)} = FIELDVALUE(cursor);
-- Move to the next sibling of the same TYPE to avoid the Title value
-- which is not an XML.Attribute
  MOVE cursor NEXTSIBLING REPEAT TYPE;
END WHILE;
```

When this ESQL is processed by the Compute node, the following output message is generated:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
</Data>
```

You can also use a SELECT statement:

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title) AS title,
          FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I);
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <title>The XML Companion</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
  <Attributes>
    <title>A Complete Guide to DB2 Universal Database</title>
    <category>Computer</category>
    <form>Paperback</form>
    <edition>2</edition>
  </Attributes>
```

```
  <Attributes>
    <title>JAVA 2 Developers Handbook</title>
    <category>Computer</category>
    <form>Hardcover</form>
    <edition>0</edition>
  </Attributes>
</Data>
```

You can qualify the SELECT with a WHERE statement to narrow down the results to obtain the same output message as the one that is generated by the WHILE statement. This second example shows that you can create the same results with less, and less complex, ESQL.

```
SET OutputRoot.XMLNS.Data.Attributes[] =
  (SELECT FIELDVALUE(I.Title.(XML.Attribute)Category) AS category,
          FIELDVALUE(I.Title.(XML.Attribute)Form) AS form,
          FIELDVALUE(I.Title.(XML.Attribute)Edition) AS edition
     FROM InputRoot.XML.Invoice.Purchases.Item[] AS I)
     WHERE I.Title = 'The XML Companion');
```

This statement generates the following output message:

```
<Data>
  <Attributes>
    <Category>Computer</Category>
    <Form>Paperback</Form>
    <Edition>2</Edition>
  </Attributes>
  </Data>
```

XML.element
This syntax element represents an XML element (a tag).

The name of the syntax element corresponds to the name of the XML element in the message. This element can have many children in the message tree, including attributes, elements, and content.

XML.tag is supported as an alternative to XML.element for compatibility with earlier versions of the product. Use XML.element in any new message flows that you create.

XML.Attribute
The XMLNS parser uses this field type for syntax elements that represent an XML attribute.

Parsing

The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

The XML.attr field type constant is also supported for compatibility with earlier versions.

XML.content
The XMLNS parser uses this syntax element to represent character data (including an XML attribute).

The name and value of the syntax element correspond to the name and value of the XML attribute that is represented. Attribute elements have no children and must always be children of an element.

Writing

When the XMLNS parser generates a bit stream from a message tree, occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe ('), within the attribute value, are replaced by the predefined XML entities &, <, >, ", and '.

XML.CDataSection
CData sections in the XML message are represented by a syntax element with field type XML.CdataSection.

The content of the CDataSection element is the value of the CDataSection element without the <![CDATA[that marks its beginning, and the]]> that marks its end.

For example, the following CData section:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

is represented by a CDataSection element with a string value of:

```
"<greeting>Hello, world!</greeting>"
```

Unlike Content, occurrences of <,>, &, ", and ' are not translated to their XML character entities (<, >, and &) when the CDataSection is produced.

When to use XML.CDataSection

A CData section is often used to embed one XML message within another. By using a CData section, you ensure that the XML reserved characters (<, >, and &) are not replaced with XML character entities.

XML.AsisElementContent also allows the production of unmodified character data, but XML.CDataSection is typically a better choice because it protects the outer message from errors in the embedded message.

Parsing the contents of a CDataSection

A common requirement is to parse the contents of a CData section to create a message tree, which you can achieve by using the ESQL statement CREATE with the PARSE clause; see "XMLNSC: Working with XML messages and bit streams" on page 1702.

XML.NamespaceDecl
A namespace declaration is represented by a syntax element with field type XML.NamespaceDecl.

Namespace declarations take one of two forms in the message tree:

- **Declaration using a namespace prefix**

  ```
  <ns1:e1 xmlns:ns1="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

Namespace	
Name	
Value	

- **Declaration of a default namespace**

  A default namespace declaration is an xmlns attribute that defines an empty prefix

  ```
  <e1 xmlns="namespace1"/>
  ```

  In the message tree, the syntax element for this namespace declaration is shown in the following table:

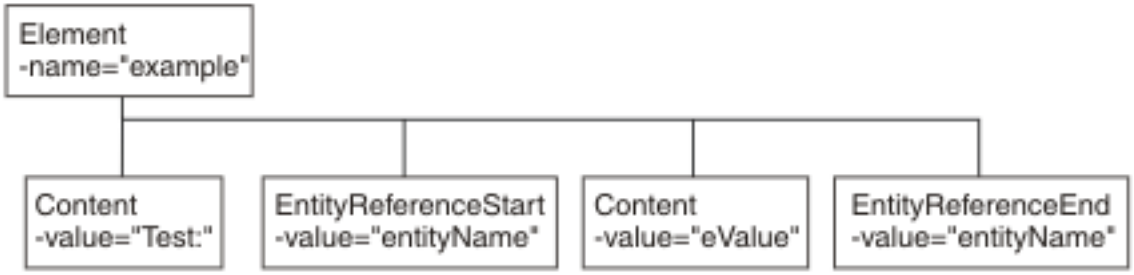
Namespace	""
Name	xmlns
Value	namespace1

  Note that, in both cases, element ‘e1' is in namespace ‘namespace1'.

XML.EntityReferenceStart and XML.EntityReferenceEnd
When an entity reference is encountered in the XML message, both the expanded form and the original entity name are stored in the syntax element tree. The name of the entity is stored as the value of the EntityReferenceStart and EntityReferenceEnd syntax elements, and any syntax elements between contain the entity expansion.

The following examples show the XML entity references in an XML document, and in tree structure form.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE example [ <!ENTITY entityName "eValue"> ]>
<example>Test: &entityName;</example>
```



The XML declaration and the document type declaration are not shown here. Refer to "XMLNS: The XML declaration" on page 1719 and "XMLNS: The DTD" on page 1720 for details of those sections of the syntax element tree.

XML.comment
An XML.comment that is encountered outside the document type declaration is represented by a syntax element with field type XML.comment. The value of the element is the comment text from the XML message.

If the value of the element contains the character sequence `-->`, the sequence is replaced with the text `-->`. This ensures that the contents of the comment cannot prematurely terminate the comment. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following is an example of the XML comment in an XML document:

```
<example><!-- This is a comment --></example>
```

XML.ProcessingInstruction
A processing instruction that is encountered outside the document type declaration is represented by a syntax element with field type XML.ProcessingInstruction.

This is a name-value element; the name of the syntax element is the processing instruction target name, and the value of the syntax element is the character data of the processing instruction. The value of the syntax element must not be empty. The name cannot be XML in either uppercase or lowercase.

If the value of the element contains the character sequence `?>`, the sequence is replaced with the text `?>`. This ensures that the content of the processing instruction cannot prematurely terminate the processing instruction. Occurrences of the following characters are not translated to their escape sequences:

```
< > & " '
```

The following shows an example of the XML processing instruction in an XML document:

```
<example><?target This is a PI.?></example>
```

XML.AsisElementContent
Use the special field type XML.AsisElementContent to precisely control generated XML.

XML.AsisElementContent is a special field type. Use the field type in a message flow to precisely control the XML that is generated in an output message, without the safeguards of the Element, Attribute, and Content syntax elements. The XMLNS parser never creates elements with this field type.

Try to avoid using AsisElementContent; there is typically a safer alternative approach. If you do use AsisElementContent, it is your responsibility to ensure that the output message is well-formed XML.

You might choose to use AsisElementContent if, for example, you want to suppress the usual behavior in which occurrences of ampersand (&), less than (<), greater than (>), double quotation mark ("), and apostrophe (') are replaced by the predefined XML entities &, <, >, ", and '.

The following example illustrates the use of AsisElementContent. The statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.Content) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message>&lt;rawMarkup&gt;</Message>
```

However, the statement:

```
Set OutputRoot.XMLNS.(XML.Element)Message.(XML.AsisElementContent) = '<rawMarkup>';
```

generates the following XML in an output message:

```
<Message><rawMarkup></Message>
```

This shows that the value of an AsisElementContent syntax element is not modified before it is written to the output message.

XML.BitStream
BitStream is a specialized element designed to aid the processing of very large messages.

XML.Bitstream is a special field type. When writing an XML message, the value of the BitStream element is written directly into the message, and the name is not important. The BitStream element might be the only element in the message tree.

The value of the element must be of type BLOB; any other data type generates an error when writing the element. Ensure that the content of the element is appropriate for use in the output message; pay special attention to the CCSID and the encoding of the XML text in the BLOB.

Use of the BitStream element is similar to the use of the AsisElementContent element, except that the AsisElementContent type converts its value into a string, whereas the BitStream element uses its BLOB value directly. This is a specialized element designed to aid the processing of very large messages.

The following ESQL excerpts demonstrate some typical use of this element. First, declare the element:

```
DECLARE StatementBitStream BLOB
```

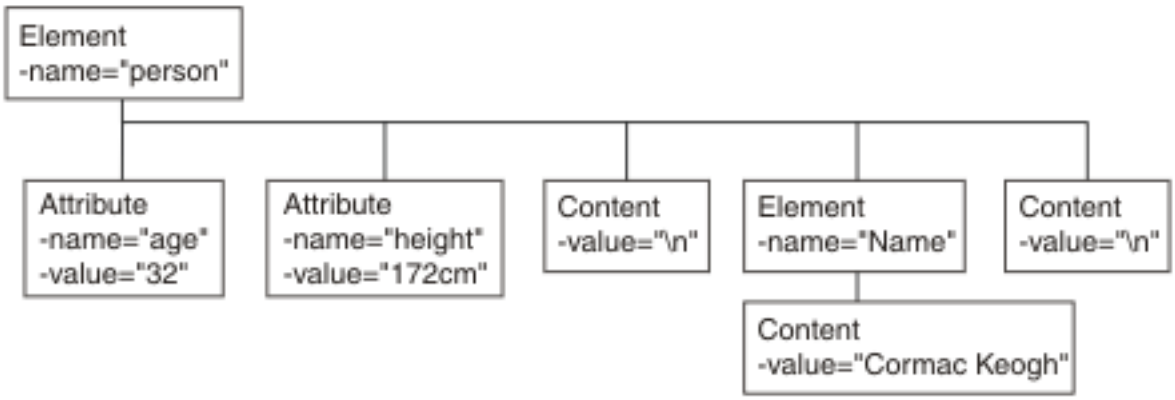
Initialize the contents of StatementBitStream from an appropriate source, such as an input message. If the source field is not of type BLOB, use the CAST statement to convert the contents to BLOB. Then create the new field in the output message; for example:

```
CREATE LASTCHILD OF resultCursor
 Type XML.BitStream
 NAME 'StatementBitStream'
 VALUE StatementBitstream;
```

XML message body example
The XMLNS parser creates a message tree that represents an XML document.

The following example shows the message tree that the XMLNS parser creates for the following snippet from a simple XML document:

```
<Person age="32" height="172cm">
<Name>Cormac Keogh</Name>
</Person>
```



Manipulating messages in the XML domain
The XML parser is like the XMLNS parser, but the XML parser has no support for namespaces or opaque parsing.

About this task

For information about how to work with the XML parser, see "Manipulating messages in the XMLNS domain" on page 1719.

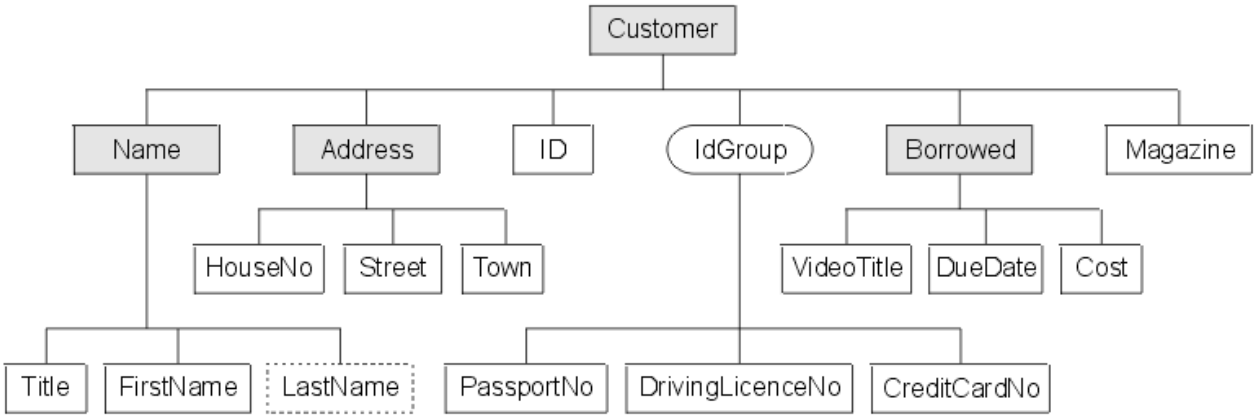
Manipulating messages in the MRM domain
How to use messages that have been modeled in the MRM domain, and that are parsed by the MRM parser.

About this task

The following topics show you how to deal with messages that have been modeled in the MRM domain, and that are parsed by the MRM parser. The physical formats associated with the message models do not affect this information unless specifically stated. Use this information in conjunction with the information about manipulating message body content; see "Manipulating message body content" on page 1632.

- "Accessing elements in a message in the MRM domain" on page 1728
- "Accessing multiple occurrences of an element in a message in the MRM domain" on page 1728
- "Accessing attributes in a message in the MRM domain" on page 1729
- "Accessing elements within groups in a message in the MRM domain" on page 1730
- "Accessing mixed content in a message in the MRM domain" on page 1732
- "Accessing embedded messages in the MRM domain" on page 1733
- "Accessing the content of a message in the MRM domain with namespace support enabled" on page 1734
- "Querying null values in a message in the MRM domain" on page 1734
- "Setting null values in a message in the MRM domain" on page 1735
- "Working with MRM messages and bit streams" on page 1736
- "Handling large MRM messages" on page 1738

The structure of a "Customer" message is shown in the following example.



The message includes a variety of structures that demonstrate how you can classify metadata to the MRM. Within an MRM message set, you can define the following objects: messages, types, groups, elements, and attributes. Folder icons that represent each of these types of objects are displayed for each message definition file in the Integration Development perspective.

Each message definition file can contribute to a namespace. You can combine several message definition files to form a complete message dictionary, which you can then deploy to an integration node.

Accessing elements in a message in the MRM domain
You can use ESQL to manipulate the logical tree that represents a message in the message flow. This topic describes how to access data for elements in a message in the MRM domain.

About this task

You can populate an element with data with the SET statement:

```
SET OutputRoot.MRM.Name = UPPER(InputRoot.MRM.Name);
```

The field reference on the left hand side of the expression refers to the element called Name within the MRM message domain. This statement takes the input value for the Name field, converts it to uppercase, and assigns the result to the same element in the output message.

The Name element is defined in the noTarget namespace. No namespace prefix is specified in front of the Name part of the field reference in the example above. If you have defined an MRM element in a namespace other than the noTarget namespace, you must also specify a namespace prefix in the statement. For example:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed.VideoTitle = 'MRM Greatest Hits';
```

For more information about using namespaces with messages in the MRM domain, see “Accessing the content of a message in the MRM domain with namespace support enabled” on page 1734.

Accessing multiple occurrences of an element in a message in the MRM domain
You can use specific ESQL code to set the value of one occurrence of an element that has multiple occurrences in a message. You can also use arrow notation to indicate the direction of search when searching for multiple occurrences of an element.

About this task

You can access MRM domain elements following the general guidance given in “Accessing known multiple occurrences of an element” on page 1636 and “Accessing unknown multiple occurrences of an element” on page 1637. Further information specific to MRM domain messages is provided in this topic.

Consider the following statements:

```
DECLARE brw NAMESPACE 'http://www.ibm.com/Borrowed';

SET OutputRoot.MRM.brw:Borrowed[1].VideoTitle = 'MRM Greatest Hits Volume 1';
SET OutputRoot.MRM.brw:Borrowed[2].VideoTitle = 'MRM Greatest Hits Volume 2';
```

The above SET statements operate on two occurrences of the element Borrowed. Each statement sets the value of the child VideoTitle. The array index indicates which occurrence of the repeating element you are interested in.

When you define child elements of a complex type (which has its `Composition` property set to `Sequence`) in a message set, you can add the same element to the complex type more than once. These instances do not have to be contiguous, but you must use the same method (array notation) to refer to them in ESQL.

For example, if you create a complex type with a `Composition` of `Sequence` that contains the following elements:

- `StringElement1`
- `IntegerElement1`
- `StringElement1`

use the following ESQL to set the value of `StringElement1`:

```
SET OutputRoot.MRM.StringElement1[1] =
              'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[2] =
              'This is the second occurrence of StringElement1';
```

You can also use the arrow notation (the greater than (>) and less than (<) symbols) to indicate the direction of search and the index to be specified:

```
SET OutputRoot.MRM.StringElement1[>] =
               'This is the first occurrence of StringElement1';
SET OutputRoot.MRM.StringElement1[<2] =
               'This is the last but one occurrence of
 StringElement1';
SET OutputRoot.MRM.StringElement1[<1] =
               'This is the last occurrence of StringElement1';
```

Refer to “Accessing known multiple occurrences of an element” on page 1636 and “Accessing unknown multiple occurrences of an element” on page 1637 for additional detail.

Accessing attributes in a message in the MRM domain
When an MRM message is parsed into a logical tree, attributes and the data that they contain are created as name-value pairs in the same way that MRM elements are. The ESQL that you code to interrogate and update the data held in attributes refers to the attributes in a similar manner.

About this task

Consider the sample MRM message described in this topic: “Manipulating messages in the MRM domain” on page 1727

The attribute LastName is defined as a child of the Name element in the Customer message. Here is an example input XML message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"

xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
            <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

The following ESQL changes the value of the LastName attribute in the output message:

```
SET OutputRoot.MRM.Name.LastName = 'Smith';
```

Be aware of the ordering of attributes when you code ESQL. When attributes are parsed, the logical tree inserts the corresponding name-value before the MRM element's child elements. In the previous example, the child elements Title and FirstName appear in the logical message tree after the attribute LastName. In the Integration Development perspective, the Outline view displays attributes after the elements. When you code ESQL to construct output messages, you must define name-value pairs for attributes before any child elements.

Accessing elements within groups in a message in the MRM domain
When an input message is parsed, structures that you have defined as groups in your message set are not represented in the logical tree, but its children are. If you want to refer to or update values for elements that are children of a groups, do not include the group in the ESQL statement. Groups do not have tags that appear in instance messages, and do not appear in user trace of the logical message tree.

About this task

Consider the following Video message:

```
<Customer xmlns:addr="http://www.ibm.com/AddressDetails"
xmlns:brw="http://www.ibm.com/BorrowedDetails">
   <Name LastName="Bloggs">
      <Title>Mr</Title>
      <FirstName>Fred</FirstName>
   </Name>
   <addr:Address>
      <HouseNo>13</HouseNo>
      <Street>Oak Street</Street>
      <Town>Southampton</Town>
   </addr:Address>
            <ID>P</ID>
   <PassportNo>J123456TT</PassportNo>
   <brw:Borrowed>
      <VideoTitle>Fast Cars</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.50</Cost>
   </brw:Borrowed>
   <brw:Borrowed>
      <VideoTitle>Cut To The Chase</VideoTitle>
      <DueDate>2003-05-23T01:00:00</DueDate>
      <Cost>3.00</Cost>
   </brw:Borrowed>
   <Magazine>0</Magazine>
</Customer>
```

When the input message is parsed, values are stored in the logical tree as shown in the following section of user trace:

```
(0x0100001B):MRM = (
  (0x01000013):Name = (
    (0x0300000B):LastName = 'Bloggs'
    (0x0300000B):Title = 'Mr'
    (0x0300000B):FirstName = 'Fred'
  )
  (0x01000013)http://www.ibm.com/AddressDetails:Address = (
    (0x0300000B):HouseNo = 13
    (0x0300000B):Street = 'Oak Street'
    (0x0300000B):Town = 'Southampton'
  )
  (0x0300000B):ID = 'P'
  (0x0300000B):PassportNo = 'J123456TT'
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Fast Cars'
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.50
  )
  (0x01000013)http://www.ibm.com/BorrowedDetails:Borrowed = (
    (0x0300000B):VideoTitle = 'Cut To The Chase '
    (0x0300000B):DueDate = TIMESTAMP '2003-05-23 00:00:00'
    (0x0300000B):Cost = 3.00
  )
  (0x0300000B):Magazine = FALSE
```

Immediately following the element named ID, the MRM message definition uses a group which has a *Composition* of Choice. The group is defined with three children: PassportNo, DrivingLicenceNo, and CreditCardNo. The choice composition dictates that instance documents must use only one of these three possible alternatives. The example shown above uses the PassportNo element.

When you refer to this element in ESQL statements, you do not specify the group to which the element belongs. For example:

```
SET OutputRoot.MRM.PassportNo = 'J999999TT';
```

If you define messages within message sets that include XML and TDS physical formats, you can determine from the message data which option of a choice has been taken, because the tags in the message represent one of the choice's options. However, if your messages have CWF physical format, or are non-tagged TDS messages, it is not clear from the message data, and the application programs processing the message must determine which option of the choice has been selected. This is known

as *unresolved choice handling*. For further information, see the description of the value of `Choice` in Message Sets: Complex type logical properties.

Accessing mixed content in a message in the MRM domain
When you define a complex type in a message model, you can optionally specify its content to be mixed. This setting, in support of mixed content in XML Schema, allows you to manipulate data that is included between elements in the message.

About this task

Consider the following example:

```
<MRM>
  <Mess1>
    abc
    <Elem1>def</Elem1>
    ghi
    <Elem2>jkl</Elem2>
    mno
    <Elem3>pqr</Elem3>
  </Mess1>
</MRM>
```

The strings abc, ghi, and mno do not represent the value of a particular element (unlike def, for example, which is the value of element Elem1). The presence of these strings means that you must model Mess1 with mixed content. You can model this XML message in the MRM using the following objects:

Message
The message *Name* property is set to `Mess1` to match the XML tag.

The *Type* property is set to `tMess1`.

Type
The complex type *Name* property is set to `tMess1`.

The *Composition* property is set to `OrderedSet`.

The complex type has mixed content.

The complex type contains the following objects:

Element
The *Name* property is set to `Elem1` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem2` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

Element
The *Name* property is set to `Elem3` to match the XML tag.

The *Type* property is set to simple type `xsd:string`.

If you code the following ESQL:

```
SET OutputRoot.MRM.*[1] = InputBody.Elem3;
SET OutputRoot.MRM.Elem1 = InputBody.*[5];
SET OutputRoot.MRM.*[3] = InputBody.Elem2;
SET OutputRoot.MRM.Elem2 = InputBody.*[3];
SET OutputRoot.MRM.*[5] = InputBody.Elem1;
SET OutputRoot.MRM.Elem3 = InputBody*[1];
```

the mixed content is successfully mapped to the following output message:

```
<MRM>
  <Mess1>
    pqr
    <Elem1>mno</Elem1>
    jkl
    <Elem2>ghi</Elem2>
    def
    <Elem3>abc</Elem3>
  </Mess1>
</MRM>
```

Accessing embedded messages in the MRM domain
If you have defined a multipart message, you have at least one message embedded within another. Within the overall complex type that represents the outer messages, you can model the inner message in two ways.

About this task

You can model the inner message in the following ways:

- An element (named E_outer1 in the following example) with its *Type* property set to a complex type that has been defined with its *Composition* property set to `Message`
- A complex type with its *Composition* property set to `Message` (named t_Embedded in the following example)

The ESQL that you need to write to manipulate the inner message varies depending on which of the above models you have used. For example, assume that you have defined:

- An outer message M_outer that has its *Type* property set to t_Outer.
- An inner message M_inner1 that has its *Type* set to t_Inner1
- An inner message M_inner2 that has its *Type* set to t_Inner2
- Type t_Outer that has its first child element named E_outer1 and its second child defined as a complex type named t_Embedded
- Type t_Embedded that has its *Composition* property set to `Message`
- Type t_Inner1 that has its first child element named E_inner11
- Type t_Inner2 that has its first child element named E_inner21
- Type t_outer1 that has its *Composition* property set to `Message`
- Element E_outer1 that has its *Type* property set to t_outer1

If you want to set the value of E_inner11, code the following ESQL:

```
SET OutputRoot.MRM.E_outer1.M_inner1.E_inner11 = 'FRED';
```

If you want to set the value of E_inner21, code the following ESQL:

```
SET OutputRoot.MRM.M_inner2.E_inner21 = 'FRED';
```

If you copy message headers from the input message to the output message, and your input message type contains a path, only the outermost name in the path is copied to the output message type.

When you configure a message flow to handle embedded messages, you can specify the path of a message type in either an MQRFH2 header (if one is present in the input message) or in the input node *Message Type* property in place of a name (for example, for the message modeled above, the path could be specified as M_Outer/M_Inner1/M_Inner2 instead of just M_Outer).

If you have specified that the input message has a physical format of either CWF or XML, any message type prefix is concatenated in front of the message type from the MQRFH2 or input node, giving a final path to use (for more information refer to “Message Sets: Multipart messages” on page 2175). The MRM uses the first item in the path as the outermost message type, then progressively works inwards when it finds a complex type with its *Composition* property set to `Message`.

If you have specified that the input message has a physical format of TDS, a different process that uses message keys is implemented. This is described in "MRM TDS format: Multipart messages" on page 2197.

For more information about path concatenations, see Message Sets: Message set properties.

Accessing the content of a message in the MRM domain with namespace support enabled
Use namespaces where appropriate for messages that are parsed by the MRM parser.

About this task

When you want to access elements of a message and namespaces are enabled, you must include the namespace when you code the ESQL reference to the element. If you do not do so, the integration node searches the `no target` namespace. If the element is not found in the `no target` namespace, the integration node searches all other known namespaces in the message dictionary (that is, within the deployed message set). For performance and integrity reasons, specify namespaces wherever they apply.

The most efficient way to refer to elements when namespaces are enabled is to define a namespace constant, and use this in the appropriate ESQL statements. This technique makes your ESQL code much easier to read and maintain.

Define a constant using the DECLARE NAMESPACE statement:

```
DECLARE ns01 NAMESPACE 'http://www.ns01.com'
.
.
SET OutputRoot.MRM.ns01:Element1 = InputBody.ns01:Element1;
```

`ns01` is interpreted correctly as a namespace because of the way that it is declared.

You can also use a CHARACTER variable to declare a namespace:

```
DECLARE ns02 CHARACTER 'http://www.ns02.com'
.
.
SET OutputRoot.MRM.{ns02}:Element2 = InputBody.{ns02}:Element2;
```

If you use this method, you must surround the declared variable with braces to ensure that it is interpreted as a namespace.

If you are concerned that a CHARACTER variable might get changed, you can use a CONSTANT CHARACTER declaration:

```
DECLARE ns03 CONSTANT CHARACTER 'http://www.ns03.com'
.
.
SET OutputRoot.MRM.{ns03}:Element3 = InputBody.{ns03}:Element3;
```

You can declare a namespace, constant, and variable within a schema, module, or function.

Querying null values in a message in the MRM domain
You can use an ESQL statement to compare an element to NULL.

About this task

If you want to compare an element to NULL, code the statement:

```
IF InputRoot.MRM.Elem2.Child1 IS NULL THEN
   DO:
    -- more ESQL --
END IF;
```

If nulls are permitted for this element, this statement tests whether the element exists in the input message, or whether it exists and contains the MRM-supplied null value. The behavior of this test depends on the physical format:

- For an XML element, if the XML tag or attribute is not in the bit stream, this test returns TRUE.

- For an XML element, if the XML tag or attribute is in the bit stream and contains the MRM null value, this test returns TRUE.
- For an XML element, if the XML tag or attribute is in the bit stream and does not contain the MRM null value, this test returns FALSE.
- For a delimited TDS element, if the element has no value between the previous delimiter and its delimiter, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a delimited TDS element, if the element has a value between the previous delimiter and its delimiter that is not the MRM-defined null value, this test returns FALSE.
- For a CWF or fixed length TDS element, if the element's value is the same as the MRM-defined null value for this element, this test returns TRUE.
- For a CWF or fixed length TDS element, if the element's value is not the same as the MRM-defined null value, this test returns FALSE.

If you want to determine if the field is missing, rather than present but with null value, you can use the ESQL CARDINALITY function.

Setting null values in a message in the MRM domain
You can use implicit or explicit null processing to set the value of an element to NULL in an output message.

About this task

To set a value of an element in an output message, you normally code an ESQL statement similar to the following one:

```
SET OutputRoot.MRM.Elem2.Child1 = 'xyz';
```

The equivalent statement is as follows:

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = 'xyz';
```

If you set the element to a non-null value, these two statements give identical results. However, if you want to set the value to null, these two statements do not give the same result:

Procedure

1. If you use the following statement to set the element to NULL, the element is deleted from the message tree:

   ```
   SET OutputRoot.MRM.Elem2.Child1 = NULL;
   ```

   The content of the output bit stream depends on the physical format:

   - For an XML element, neither the XML tag or attribute nor its value are included in the output bit stream.
   - For a Delimited TDS element, neither the tag (if appropriate) nor its value are included in the output bit stream. The absence of the element is typically conveyed by two adjacent delimiters.
   - For a CWF or Fixed Length TDS element, the content of the output bit stream depends on whether you set the `Default Value` property for the element. If you set this property, the default value is included in the bit stream. If you did not set the property, an exception is raised.

   This behavior is called implicit null processing.

2. If you set the value of this element to NULL as follows, the element is not deleted from the message tree. Instead, a special value of NULL is assigned to the element. The content of the output bit stream depends on the settings of the physical format null-handling properties.

```
SET OutputRoot.MRM.Elem2.Child1 VALUE = NULL;
```

   This behavior is called explicit null processing.

Results

Setting a complex element to NULL deletes that element and all its child elements.

Working with MRM messages and bit streams
When you use the ASBITSTREAM function or the CREATE FIELD statement with a PARSE clause you must consider various restrictions.

About this task

The ASBITSTREAM function

If you code the ASBITSTREAM function with the parser mode option set to *RootBitStream*, to parse a message tree to a bit stream, the result is an MRM document in the format specified by the message format that is built from the children of the target element in the normal way.

The target element must be a predefined message defined within the message set, or can be a self-defined message if you are using an XML physical format. This algorithm is identical to that used to generate the normal output bit stream. A well-formed bit stream obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause.

If you code the ASBITSTREAM function with the parser mode option set to *FolderBitStream*, to parse a message tree to a bit stream, the generated bit stream is an MRM element built from the target element and its children. Unlike *RootBitStream* mode the target element does not have to represent a message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message, the path from the message to the target element within the message must be specified in the *Message Type*. The format of the path is the same as that used by message paths except that the message type prefix is not used.

For example, suppose the following message structure is used:

```
          Message
              elem1
                   elem11
                   elem12
```

To serialize the subtree representing element `elem12` and its children, specify the message path `'message/elem1/elem12'` in the *Message Type*.

If an element in the path is qualified by a namespace, specify the namespace URI between {} characters in the message path. For example if element `elem1` is qualified by namespace `'http://www.ibm.com/temp'`, specify the message path as `'message/{http://www.ibm.com/temp}elem1/elem12'`

This mode can be used to obtain a bit stream description of arbitrary sub-trees owned by an MRM parser. When in this mode, with a physical format of XML, the XML bit stream generated is not enclosed by the 'Root Tag Name' specified for the Message in the Message Set. No XML declaration is created, even if not suppressed in the message set properties.

Bit streams obtained in this way can be used to re-create the original tree by using a CREATE statement with a PARSE clause (by using a mode of *FolderBitStream*).

The CREATE statement with a PARSE clause

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *RootBitStream*, to parse a bit stream to a message tree, the expected bit stream is a normal MRM document. A field in the

tree is created for each field in the document. This algorithm is identical to that used when parsing a bit stream from an input node

If you code a CREATE statement with a PARSE clause, with the parser mode option set to *FolderBitStream*, to parse a bit stream to a message tree, the expected bit stream is a document in the format specified by the Message Format, which is either specified directly or inherited. Unlike *RootBitStream* mode the root of the document does not have to represent an MRM message; it can represent a predefined element within a message or self-defined element within a message.

So that the MRM parser can correctly parse the message the path from the message to the target element within the message must be specified in the *Message Type*. The format of the message path is the same as that used for the ASBITSTREAM function described above.

Example of using the ASBITSTREAM function and CREATE statement with a PARSE clause in FolderBitStream mode

The following ESQL uses the message definition described above. The ESQL serializes part of the input tree by using the ASBITSTREAM function, then uses the CREATE statement with a PARSE clause to re-create the subtree in the output tree. The Input message and corresponding Output message are shown below the ESQL.

```
CREATE COMPUTE MODULE DocSampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();

  -- Set the options to be used by ASBITSTREAM and CREATE ... PARSE
  -- to be FolderBitStream and enable validation
  DECLARE parseOptions INTEGER BITOR(FolderBitStream, ValidateContent,
                       ValidateValue, ValidateLocalError);

  -- Serialise the elem12 element and its children from the input bitstream
  -- into a variable
       DECLARE subBitStream BLOB
         ASBITSTREAM(InputRoot.MRM.elem1.elem12
           OPTIONS parseOptions
           SET 'DocSample'
           TYPE 'message/elem1/elem12'
           FORMAT 'XML1');

       -- Set the value of the first element in the output tree
       SET OutputRoot.MRM.elem1.elem11 = 'val11';

       -- Parse the serialized sub-tree into the output tree
       IF subBitStream IS NOT NULL THEN
           CREATE LASTCHILD OF OutputRoot.MRM.elem1
               PARSE ( subBitStream
                       OPTIONS parseOptions
                       SET 'DocSample'
                       TYPE 'message/elem1/elem12'
                       FORMAT 'XML1');
       END IF;

  -- Convert the children of elem12 in the output tree to uppercase
  SET OutputRoot.MRM.elem1.elem12.elem121 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem121);

  SET OutputRoot.MRM.elem1.elem12.elem122 =
   UCASE(OutputRoot.MRM.elem1.elem12.elem122);

       -- Set the value of the last element in the output tree
       SET OutputRoot.MRM.elem1.elem13 = 'val13';

  RETURN TRUE;
 END;

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

```
END MODULE;
```

Input message :

```
<message>
    <elem1>
        <elem11>value11</elem11>
        <elem12>
            <elem121>value121</elem121>
            <elem122>value122</elem122>
        </elem12>
        <elem13>value13</elem13>
    </elem1>
</message>
```

Output message :

```
<message>
    <elem1>
        <elem11>val11</elem11>
        <elem12>
            <elem121>VALUE121</elem121>
            <elem122>VALUE122</elem122>
        </elem12>
        <elem13>val13</elem13>
    </elem1
</message
```

Handling large MRM messages
When an input bit stream is parsed, and a logical tree is created, the tree representation of an MRM message is typically larger, and in some cases much larger, than the corresponding bit stream.

About this task

The reasons for this large size include:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode that can double the original size
- The inclusion of field names that can be contained implicitly within the bit stream
- The presence of control data that is associated with the integration node's operation

Manipulation of a large message tree can, therefore, demand a great deal of storage. If you design a message flow that handles large messages that are made up of repeating structures, you can code specific ESQL statements that help to reduce the storage load on the integration node. These statements support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These ESQL statements cause the integration node to perform limited parsing of the message, and to keep only that part of the message tree that reflects a single record in storage at a time. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain ESQL variables, or you can save values in another part of the message tree, for example LocalEnvironment.

This technique reduces the memory that is used by the integration node to that needed to hold the full input and output bit streams, plus that required for one record's trees. It provides memory savings when even a few repeats are encountered in the message. The integration node makes use of partial parsing, and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your Compute node apply these general techniques:

- Copy the body of the input message as a bit stream to a special folder in the output message. This creates a modifiable copy of the input message that is not parsed and therefore uses a minimum amount of memory.
- Avoid any inspection of the input message; this avoids the need to parse the message.

- Use a loop and a reference variable to step through the message one record at a time. For each record:
  - Use normal transforms to build a corresponding output subtree in a second special folder.
  - Use the ASBITSTREAM function to generate a bit stream for the output subtree that is stored in a *BitStream* element, placed in the position in the tree, that corresponds to its required position in the final bit stream.
  - Use the DELETE statement to delete both the current input and the output record message trees when you complete their manipulation.
  - When you complete the processing of all records, detach the special folders so that they do not appear in the output bit stream.

You can vary these techniques to suit the processing that is required for your messages. The following ESQL provides an example of one implementation.

The ESQL is dependent on a message set called `LargeMessageExanple` that has been created to define messages for both the Invoice input format and the Statement output format. A message called `AllInvoices` has been created that contains a global element called `Invoice` that can repeat one or more times, and a message called `Data` that contains a global element called `Statement` that can repeat one or more times.

The definitions of the elements and attributes have been given the correct data types, therefore, the CAST statements used by the ESQL in the XML example are no longer required. An XML physical format with name `XML1` has been created in the message set which allows an XML message corresponding to these messages to be parsed by the MRM.

When the `Statement` tree is serialized using the ASBITSTREAM function the *Message Set*, *Message Type*, and *Message Format* are specified as parameters. The *Message Type* parameter contains the path from the message to the element being serialized which, in this case, is `Data/Statement` because the `Statement` element is a direct child of the `Data` message.

The input message to the flow is the same Invoice example message used in other parts of the documentation except that it is contained between the tags:

```
       <AllInvoices> ....  </AllInvoices>
```

Example

```
CREATE COMPUTE MODULE LargeMessageExampleFlow_Compute
 CREATE FUNCTION Main() RETURNS BOOLEAN
 BEGIN
  CALL CopyMessageHeaders();
    -- Create a special folder in the output message to hold the input tree
    -- Note : SourceMessageTree is the root element of an MRM parser
    CREATE LASTCHILD OF OutputRoot.MRM DOMAIN 'MRM' NAME 'SourceMessageTree';

    -- Copy the input message to a special folder in the output message
    -- Note : This is a root to root copy which will therefore not build trees
    SET OutputRoot.MRM.SourceMessageTree = InputRoot.MRM;

    -- Create a special folder in the output message to hold the output tree
    CREATE FIELD OutputRoot.MRM.TargetMessageTree;

    -- Prepare to loop through the purchased items
    DECLARE sourceCursor REFERENCE TO OutputRoot.MRM.SourceMessageTree.Invoice;
    DECLARE targetCursor REFERENCE TO OutputRoot.MRM.TargetMessageTree;
    DECLARE resultCursor REFERENCE TO OutputRoot.MRM;
    DECLARE grandTotal   FLOAT     0.0e0;

    -- Create a block so that it's easy to abandon processing
    ProcessInvoice: BEGIN
     -- If there are no Invoices in the input message, there is nothing to do
     IF NOT LASTMOVE(sourceCursor) THEN
        LEAVE ProcessInvoice;
     END IF;

     -- Loop through the invoices in the source tree
     InvoiceLoop : LOOP
        -- Inspect the current invoice and create a matching Statement
        SET targetCursor.Statement =
```

```
        THE (
           SELECT
             'Monthly'                        AS Type,
             'Full'                           AS Style,
             I.Customer.FirstName             AS Customer.Name,
             I.Customer.LastName              AS Customer.Surname,
             I.Customer.Title                 AS Customer.Title,
             (SELECT
                FIELDVALUE(II.Title)    AS Title,
                II.UnitPrice * 1.6    AS Cost,
                II.Quantity                   AS Qty
             FROM I.Purchases.Item[] AS II
             WHERE II.UnitPrice> 0.0               ) AS Purchases.Article[],
             (SELECT
                SUM( II.UnitPrice *
                        II.Quantity  *
                     1.6                          )
              FROM I.Purchases.Item[] AS II                    ) AS Amount,
              'Dollars'                                  AS Amount.Currency
            FROM sourceCursor AS I
            WHERE I.Customer.LastName <> 'White'
          );

      -- Turn the current Statement into a bit stream
      -- The SET parameter is set to the name of the message set
       -- containing the MRM definition
      -- The TYPE parameter contains the path from the from the message
       -- to element being serialized
      -- The FORMAT parameter contains the name of the physical format
       -- name defined in the message
      DECLARE StatementBitStream BLOB
        ASBITSTREAM(targetCursor.Statement
         OPTIONS FolderBitStream
         SET 'LargeMessageExample'
         TYPE 'Data/Statement'
         FORMAT 'XML1');

      -- If the SELECT produced a result (that is, it was not filtered
       -- out by the WHERE clause), process the Statement
      IF StatementBitStream IS NOT NULL THEN
        -- create a field to hold the bit stream in the result tree
        -- The Type of the element is set to MRM.BitStream to indicate
         -- to the MRM Parser that this is a bitstream
        CREATE LASTCHILD OF resultCursor
           Type  MRM.BitStream
           NAME  'Statement'
           VALUE StatementBitStream;

        -- Add the current Statement's Amount to the grand total
        SET grandTotal = grandTotal + targetCursor.Statement.Amount;
      END IF;

      -- Delete the real Statement tree leaving only the bit stream version
      DELETE FIELD targetCursor.Statement;

      -- Step onto the next Invoice, removing the previous invoice and any
      -- text elements that might have been interspersed with the Invoices
      REPEAT
        MOVE sourceCursor NEXTSIBLING;
        DELETE PREVIOUSSIBLING OF sourceCursor;
      UNTIL (FIELDNAME(sourceCursor) = 'Invoice')
             OR (LASTMOVE(sourceCursor) = FALSE)
      END REPEAT;

      -- If there are no more invoices to process, abandon the loop
      IF NOT LASTMOVE(sourceCursor) THEN
        LEAVE InvoiceLoop;
      END IF;

   END LOOP InvoiceLoop;
  END ProcessInvoice;

  -- Remove the temporary source and target folders
  DELETE FIELD OutputRoot.MRM.SourceMessageTree;
  DELETE FIELD OutputRoot.MRM.TargetMessageTree;

  -- Finally add the grand total
  SET resultCursor.GrandTotal = grandTotal;

  -- Set the output MessageType property to be 'Data'
  SET OutputRoot.Properties.MessageType = 'Data';
```

```
  RETURN TRUE;
 END;

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

END MODULE;
```

Manipulating messages in the JMS domains
This topic provides information specific to dealing with messages that belong to the JMSMap and JMSStream domains. These messages are parsed by the generic XML parser.

About this task

Messages that belong to the JMS domains are processed by the XML parser, so you can follow the guidance provided for XML messages in "Manipulating messages in the XML domain" on page 1727, in conjunction with the information in "Manipulating message body content" on page 1632.

The JMSMap and JMSStream domains support MapMessage and StreamMessage messages. Other kinds of JMS message are supported by other domains. For further information about using JMS messages with IBM App Connect Enterprise, see "Using JMS in IBM App Connect Enterprise" on page 852.

Manipulating messages in the IDOC domain
Use ESQL from a Compute node to copy the incoming IDoc to the outgoing IDoc, and manipulate the message.

About this task

Note: The IDOC domain is deprecated and is not recommended for developing new message flows. Instead use the MRM domain with a TDS physical format. See "MRM parser and domain" on page 545.

A valid IDoc message flows out of SAP and is sent to the MQSeries link for R/3.

When this IDoc has been committed successfully to the outbound IBM MQ queue, the input node of the message flow reads it from that queue and generates the syntax element tree.

The Compute node manipulates this syntax element tree and, when it has finished, passes the output message to subsequent nodes in the message flow. When the message reaches the output node, the IDOC parser is called to rebuild the bit stream from the tree.

The message flow must create an output message in a similar format to the input message.

See Field names of the IDOC parser structures for the field names in the DC (Control Structure) and DD (Data Structure) that are recognized by the IDOC parser

Procedure

Use the following ESQL example from a Compute node:

```
SET OutputRoot = InputRoot;
SET OutputRoot.IDOC.DC[1].tabnam = 'EDI_DC40  ';
SET OutputRoot.IDOC.DD[2].sdatatag.MRM.maktx = 'Buzzing all day';
```

Results

The first line of the code copies the incoming IDoc to the outgoing IDoc.

The second line sets the *tabname* of the first DC.

The third line uses the second DD segment, which in this example is of type E2MAKTM001, and sets the *maktx* field.

Accessing fields of the IDoc using ESQL

About this task

Use the ESQL editor Content Assist to complete the SAP-defined fields of the IDoc.

After the sdatatag tag in an ESQL statement, the next tag is MRM, which you must enter manually, followed by the field name that is to be manipulated. Specify the name of the field within the message segment here, not the name of the message segment.

For example, the following code sets the segment name of the IDoc:

```
SET OutputRoot.IDOC.DD[I].segnam  = 'E2MAKTM001';
```

The following example sets the msgfn field within the E2MAKTM001 segment:

```
SET OutputRoot.IDOC.DD[I].sdatatag.MRM.msgfn = '006';
```

Manipulating messages in the MIME domain

A MIME message does not need to be received over a particular transport. For example, a message can be received over HTTP by using an HTTPInput node, or over IBM MQ by using an MQInput node. The MIME parser is used to process a message if the message domain is set to MIME in the input node properties, or if you are using IBM MQ, and the MQRFH2 header has a message domain of MIME.

This topic explains how to deal with messages that belong to the MIME domain, and are parsed by the MIME parser. Use this information in conjunction with the information in "Manipulating message body content" on page 1632.

You can manipulate the logical tree using ESQL before passing the message on to other nodes in the message flow. A message flow can also create a MIME domain tree using ESQL. When a MIME domain message reaches an output node, the MIME parser is called to rebuild the bit stream from the logical tree.

The following examples show how to manipulate MIME messages:

- "Creating a new MIME tree" on page 1742
- "Modifying an existing MIME tree" on page 1743
- "Managing Content-Type" on page 1743

Creating a new MIME tree

A message flow often receives, modifies, and returns a MIME message. In this case, you can work with the valid MIME tree that is created when the input message is parsed. If a message flow receives input from another domain, such as XMLNS, and returns a MIME message, you must create a valid MIME tree. Use the following ESQL example in a Compute node to create the top-level structure for a single-part MIME tree:

```
CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Data';
```

The message flow must also ensure that the MIME Content-Type is set correctly, as explained in "Managing Content-Type" on page 1743. The flow must then add the message data into the MIME tree. The following ESQL examples show how you can do this. In each case, a Data element is created with the domain BLOB.

- A bit stream from another part of the tree is used. This example shows how a bit stream could be created from an XML message that is received by the message flow. The flow then invokes the BLOB parser to store the data under the Data element.

  ```
  DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
  ```

```
CREATE LASTCHILD OF M.Data DOMAIN('BLOB') PARSE(partData);
```

- Instead of parsing the bit stream, create the new structure, then attach the data to it, as shown in this ESQL example:

```
DECLARE partData BLOB ASBITSTREAM(InputRoot.XMLNS);
CREATE LASTCHILD OF M.Data DOMAIN('BLOB') NAME 'BLOB';
CREATE LASTCHILD OF M.Data.BLOB NAME 'BLOB' VALUE partData;
```

Both of these approaches create the same tree structure. The first approach is better because explicit knowledge of the tree structure that the BLOB parser requires is not built into the flow.

More commonly, the Compute node must build a tree for a multipart MIME document. The following ESQL example shows how you can do this, including setting the top-level Content-Type property.

```
DECLARE part1Data BLOB ASBITSTREAM(InputRoot.XMLNS, InputProperties.Encoding,
 InputProperties.CodedCharSetId);

SET OutputRoot.Properties.ContentType = 'multipart/related; boundary=myBoundary';

CREATE FIELD OutputRoot.MIME TYPE Name;
DECLARE M REFERENCE TO OutputRoot.MIME;
CREATE LASTCHILD OF M TYPE Name NAME 'Parts';
CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P1 REFERENCE TO M.Parts.Part[1];
CREATE FIELD P1."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P1."Content-Id"   TYPE NameValue VALUE 'part one';
CREATE LASTCHILD OF P1 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P1.Data DOMAIN('BLOB') PARSE(part1Data);

CREATE LASTCHILD OF M.Parts TYPE Name NAME 'Part';
DECLARE P2 REFERENCE TO M.Parts.Part[2];
CREATE FIELD P2."Content-Type" TYPE NameValue VALUE 'text/plain';
CREATE FIELD P2."Content-Id"   TYPE NameValue VALUE 'part two';
CREATE LASTCHILD OF P2 TYPE Name NAME 'Data';
CREATE LASTCHILD OF P2.Data DOMAIN('BLOB') PARSE(part2Data);
```

Modifying an existing MIME tree

This ESQL example adds a new MIME part to an existing multipart MIME message. If the message is not multipart, it is not modified.

```
SET OutputRoot = InputRoot;

-- Check to see if the MIME message is multipart or not.
IF LOWER(InputProperties.ContentType) LIKE 'multipart/%'
THEN
  CREATE LASTCHILD OF OutputRoot.MIME.Parts NAME 'Part';

  DECLARE P REFERENCE TO OutputRoot.MIME.Parts.[<];
  CREATE FIELD P."Content-Type" TYPE NameValue VALUE 'text/xml';
  CREATE FIELD P."Content-ID"   TYPE NameValue VALUE 'new part';
  CREATE LASTCHILD OF P TYPE Name NAME 'Data';

  -- This is an artificial way of creating some BLOB data.
  DECLARE newBlob BLOB '4f6e652074776f2074687265650d0a';
  CREATE LASTCHILD OF P.Data DOMAIN('BLOB') PARSE(newBlob);
END IF;
```

If you receive a MIME message, for example; through an EmailInput node, and you know the format of your message, you might want to reparse the message. For example:

```
CREATE LASTCHILD OF OutputRoot.XMLNSC.emailData  DOMAIN('XMLNSC')
PARSE(InputRoot.MIME.Data.BLOB.BLOB,InputProperties.Encoding,
InputProperties.CodedCharSetId);
```

Managing Content-Type

When you create a new MIME message tree, or when you modify the value of the MIME boundary string, make sure that the MIME Content-Type header is set correctly by setting the ContentType value in the

integration node Properties subtree. The following example shows how to set the ContentType value for a MIME part with simple content:

```
SET OutputRoot.Properties.ContentType = 'text/plain';
```

Do not set the Content-Type value directly in the MIME tree or HTTP trees because the value is ignored or used inconsistently.

When you receive a MIME message, you can filter or route the message content based on the content-Type. The following example shows an XPath query that can be used in a Route node to filter the content based on whether the message contains an attachment or not:

```
starts-with($Root/MIME/Content-Type,"multipart")
```

Manipulating messages in the BLOB domain
How to deal with messages that belong to the BLOB domain, and that are parsed by the BLOB parser.

About this task

You cannot manipulate the contents of a BLOB message, because it has no predefined structure. However, you can refer to its contents using its known position within the bit stream, and process the message with a minimum of knowledge about its contents.

The BLOB message body parser does not create a tree structure in the same way that other message body parsers do. It has a root element BLOB, that has a child element, also called BLOB, that contains the data.

You can refer to message content using substrings if you know the location of a particular piece of information within the BLOB data. For example, the following expression identifies the tenth byte of the message body:

```
InputBody.BLOB.BLOB[10]
```

The following expression references 10 bytes of the message data starting at offset 10:

```
SUBSTRING(InputBody.BLOB.BLOB from 10 for 10)
```

You can use the Mapping node to map to and from a predefined BLOB message, and to map to and from items of BLOB data.

Simple example to write a string in the output message

About this task

The following simple example allows you to write some character data in your ESQL (for example, if you have read some character fields from a database) out as a BLOB:

```
CALL CopyMessageHeaders();
 -- CALL CopyEntireMessage();
 DECLARE mystring CHARACTER;
 SET mystring='hello';
 SET OutputRoot.BLOB.BLOB=CAST (mystring AS BLOB CCSID 1208);
```

Manipulating messages in the JSON domain
You can use the Graphical Data Mapping editor, Java, or ESQL to manipulate JSON messages.

You can create JSON messages that contain JSON objects, JSON arrays, or both, by creating elements in the logical message tree, under the Data element that is owned by the JSON parser root. For more information, see “Creating a JSON message” on page 1745.

You can modify JSON messages. For more information, see “Modifying a JSON message” on page 1746.

Use this information in conjunction with the information in “Manipulating message body content” on page 1632.

Creating a JSON message
You can create JSON message data that contains JSON objects, JSON arrays, or both, by creating elements in the logical message tree, under the `Data` element that is owned by the JSON parser root.

A JSON message can have either an anonymous object or an anonymous array as the root of the data. It is not possible to put a JSON string literal, number, boolean, or null value as the root of the data using the JSON parser. When you create a JSON array in the logical message tree, the JSON array name is placed in a tree element that has a type that is set to the JSON parser element type `JSON.Array`.

The items in the JSON array are placed in the logical tree as `NameValue` elements, as children of the `JSON.Array` element, with the required value. The names of these array item elements are not used by the JSON serializer because JSON array items are anonymous. However, for consistency with the name that is used by the JSON parser, use the name `Item` when you define the array item elements.

Creating a JSON object message

The following example shows how to create a JSON message that is formatted with an object at the root level, with the form `{ --- }`.

This example shows how to create a simple JSON object message with one name-value pair:

```
{"Message":"Hello World"}
```

The following ESQL code can be used to create the message:

```
SET OutputRoot.JSON.Data.Message = 'Hello World';
```

The following Java code can also be used to create the message:

```
MbElement outRoot =
   outMessage.getRootElement();
MbElement outJsonRoot =
   outRoot.createElementAsLastChild(MbJSON.PARSER_NAME);
MbElement outJsonData =
   outJsonRoot.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.DATA_ELEMENT_NAME, null);
MbElement outJsonTest =
   outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Message", "Hello World");
```

Creating a JSON array message

The following example shows how to create a message that is formatted with an array at the root level, in the form `[ --- ]`.

This example shows how to create a JSON array message:

```
["valueA","valueB"]
```

The following ESQL code can also be used to create the array:

```
CREATE FIELD OutputRoot.JSON.Data IDENTITY (JSON.Array)Data;
CREATE LASTCHILD OF OutputRoot.JSON.Data TYPE NameValue NAME 'Item' VALUE 'valueA';
CREATE LASTCHILD OF OutputRoot.JSON.Data TYPE NameValue NAME 'Item' VALUE 'valueB';
```

The following Java code can also be used to create the array:

```
MbElement outJsonRoot =
   outRoot.createElementAsLastChild("JSON");
MbElement outJsonData =
   outJsonRoot.createElementAsLastChild(MbJSON.ARRAY, "Data", null);
   outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
                                        "Item", "valueA");
   outJsonData.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
                                        "Item", "valueB");
```

The message is created in the integration node logical message tree in the following form:

```
Message: ( ['json' : 0xc552990]
```

```
  (0x01001000:Array     ):Data    = (
    (0x03000000:NameValue):Item = 'valueA' (CHARACTER)
    (0x03000000:NameValue):Item = 'valueB' (CHARACTER)
 )
```

Creating a JSON object array message

The following example shows how to create a message that is formatted with an array of objects at the root level, in the form `[{--},{--},...]`.

This example shows how to create the JSON object array message:

```
[{"Nam1":"val1","Num1":1},{"Nam2":"val2","Num2":2}]
```

The following ESQL code can also be used to create the array:

```
CREATE FIELD OutputRoot.JSON.Data IDENTITY(JSON.Array)Data;
SET OutputRoot.JSON.Data.Item[1].Nam1 = 'val1';
SET OutputRoot.JSON.Data.Item[1].Num1 = 1;
SET OutputRoot.JSON.Data.Item[2].Nam2 = 'val2';
SET OutputRoot.JSON.Data.Item[2].Num2 = 2;
```

The following Java code can also be used to create the array:

```
MbElement jsonData =

 outMessage.getRootElement().createElementAsLastChild(MbJSON.PARSER_NAME).createElementAsLastChild
                                        (MbJSON.ARRAY,MbJSON.DATA_ELEMENT_NAME, null);
MbElement jsonFirstArrayItem =
   jsonData.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.ARRAY_ITEM_NAME, null);
   jsonFirstArrayItem.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Nam1", "val1");
   jsonFirstArrayItem.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Num1", new
 Integer(1));

MbElement jsonSecondArrayItem =
   jsonData.createElementAsLastChild(MbElement.TYPE_NAME, MbJSON.ARRAY_ITEM_NAME, null);
   jsonSecondArrayItem.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Nam2", "val2");
   jsonSecondArrayItem.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "Num2", new
 Integer(2));
```

The message is created in the integration node logical message tree in the following form:

```
Message: ( ['json' : 0xc673900]
  (0x01001000:Array):Data = (
    (0x01000000:Object):Item = (
      (0x03000000:NameValue):nam1 = 'val1' (CHARACTER)
      (0x03000000:NameValue):num1 = 1 (INTEGER)
    )(0x01000000:Object):Item = (
      (0x03000000:NameValue):nam2 = 'val2' (CHARACTER)
      (0x03000000:NameValue):num2 = 2 (INTEGER)
    )
  )
)
```

Modifying a JSON message
You can modify JSON objects and JSON arrays.

JSON data streams are parsed into the logical message tree and placed under the `Data` element that is owned by the JSON parser root. The JSON data objects and arrays can be accessed and modified from each supported language as follows:

- ESQL as `OutputRoot.JSON.Data.`*`path to required object or array`*
- Java as `/JSON/Data/`*`path to required object or array`*

You can also manipulate JSON messages in the Graphical Data Mapping editor. For more information, see Creating or transforming a JSON output message by using a message map.

The following example shows a possible JSON message:

```
{
    "name" : "John Doe",
```

```
    "age" : -1,
    "known" : false,
    "address" : { "street" : null,
                  "city" : "unknown" },
    "belongings" : ["this", "that", "the other"]
}
```

The JSON parser parses the input JSON bit stream, to produce the following integration node logical message tree:

```
(Message):JSON     = ( ['json' : 0xhhhhhh]
    (0x01000000:Object):Data      = (
      (0x03000000:NameValue): name    = 'John Doe' (CHARACTER)
      (0x03000000:NameValue): age     = -1 (INTEGER)
      (0x03000000:NameValue): known   = FALSE (BOOLEAN)
      (0x01000000:Object     ): address = (
        (0x03000000:NameValue): street = NULL (UNKNOWN)
        (0x03000000:NameValue): city = 'unknown' (CHARACTER)
      )
      (0x01001000:Array     ): belongings = (
        (0x03000000:NameValue): Item = 'this' (CHARACTER)
        (0x03000000:NameValue): Item = 'that' (CHARACTER)
        (0x03000000:NameValue): Item = 'the other' (CHARACTER)
      )
    )
  )
```

This message tree can be modified through ESQL as:

```
SET OutputRoot.JSON.Data.age =  InputRoot.JSON.Data.age + 22;   -- Set age to 21
SET OutputRoot.JSON.Data.belongings.Item[4] = 'an other';
SET OutputRoot.JSON.Data.belongings.Item[5] = 'and another';
```

The message tree can be modified through Java as:

```
MbElement ageEl = message.getRootElement().getLastChild().getFirstElementByPath("/JSON/Data/
age");
int age = ((Integer)ageEl.getValue()).intValue();
ageEl.setValue(age + 22);       // Set age to 21
inMessage.getRootElement().getLastChild().getFirstElementByPath("/JSON/Data/belongings/
Item[3]").setValue('an other');
```

JSON with a multidimensional array

The following example shows JSON input containing a multidimensional array:

```
{
    "customer" : "Joe",
    "orders" : [ [ "thing1", 1, 10.1 ],
                 [ "thing2", 2, 20.2 ]  ]
}
```

The following integration node message tree is produced:

```
(Message):JSON   = ( ['json' : 0xhhhhhh]
    (0x01000000:Object):Data   = (
      (0x03000000:NameValue):customer = 'Joe' (CHARACTER)
      (0x01001000:Array):orders = (
        (0x01001000:Array):Item   = (
          (0x03000000:NameValue):Item = 'thing1' (CHARACTER)
          (0x03000000:NameValue):Item = 1 (INTEGER)
          (0x03000000:NameValue):Item = 1.01E+1 (FLOAT)
        )
        (0x01001000:Array):Item   = (
          (0x03000000:NameValue):Item = 'thing2' (CHARACTER)
          (0x03000000:NameValue):Item = 2 (INTEGER)
          (0x03000000:NameValue):Item = 2.02E+1 (FLOAT)
        )
      )
    )
  )
```

This message tree is accessed through ESQL in the following way (you can use either the name *Item* or an asterisk (*) as a wildcard):

```
InputRoot.JSON.Data.orders.Item[1].Item[1]   -- 'thing1'
InputRoot.JSON.Data.orders.*[2].*[3]         -- 2.02E+1
```

The message tree is accessed through Java in the following way (you can use either the name *Item*, which the JSON parser gives to array items, or an asterisk (*) as a wildcard):

```
inMessage.getRootElement().getFirstElementByPath("/JSON/Data/orders/Item[1]/Item[1]");  //
 'thing1'
inMessage.getRootElement().getFirstElementByPath("/JSON/Data/orders/*[2]/*[3]");  // '2.02'
```

Using the CALL statement to call a user-written routine
The ESQL CALL statement calls routines that have been created and implemented in different ways.

About this task

A routine is a user-defined function or procedure that has been defined by one of the following statements:

- CREATE FUNCTION
- CREATE PROCEDURE

You can use the CALL statement to call a routine that has been implemented in any of the following ways:

- ESQL
- Java
- As a stored procedure in a database
- As a function that is provided by IBM App Connect Enterprise

  You can use CALL to call built-in functions and user-defined SQL functions, but typically you use their names directly in expressions .

For details of the syntax and parameters of the CALL statement, see CALL statement. For an example of the use of CALL, see the examples in CREATE PROCEDURE statement.

Calling an ESQL routine

About this task

A routine is called as an ESQL method if the routine's definition specifies a LANGUAGE clause of ESQL or if the routine is a built-in function. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. An ESQL routine is allowed to return any ESQL data type, excluding List and Row.

Calling a Java routine

About this task

A routine is called as a Java method if the routine's definition specifies a LANGUAGE clause of JAVA. An exact one-to-one matching of the data types and directions of each parameter, between the definition and the CALL, is required. If the Java method has a void return type, the INTO clause cannot be used because no value exists to return.

A Java routine can return any data type in the ESQL-to-Java data-type mapping table, excluding List and Row.

Calling a database stored procedure

About this task

A routine is called as a database stored procedure if the routine's definition has a LANGUAGE clause of DATABASE.

When a call is made to a database stored procedure, the integration node searches for a definition (created by a CREATE PROCEDURE statement) that matches the procedure's local name. The integration node then uses the following sequence to resolve the name by which the procedure is known in the database and the database schema to which it belongs:

1. If the CALL statement specifies an IN clause, the name of the data source, the database schema, or both, is taken from the IN clause.
2. If the name of the data source is not provided by an IN clause on the CALL statement, it is taken from the DATASOURCE attribute of the node.
3. If the database schema is not provided by an IN clause on the CALL statement, but is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, it is taken from the EXTERNAL NAME clause.
4. If no database schema is specified on the EXTERNAL NAME clause of the CREATE PROCEDURE statement, the database's user name is used as the schema name. If a matching procedure is found, the routine is called.

The chief use of the CALL statement's IN clause is that it allows the data source, the database schema, or both, to be chosen dynamically at run time. (The EXTERNAL SCHEMA clause also allows the database schema which contains the stored procedure to be chosen dynamically, but it is not as flexible as the IN clause and is retained only for compatibility with earlier versions. Its use in new applications is deprecated.)

If the called routine has any DYNAMIC RESULT SETS specified in its definition, the number of expressions in the CALL statement's *ParameterList* must match the number of parameters to the routine, plus the number of DYNAMIC RESULT SETS. For example, if the routine has three parameters and two DYNAMIC RESULT SETS, the CALL statement must pass five parameters to the called routine. The parameters passed for the two DYNAMIC RESULT SETS must be list parameters; that is, they must be field references qualified with array brackets [ ]; for example, Environment.ResultSet1[].

A database stored procedure is allowed to return any ESQL data type, excluding Interval, List, and Row.

Configuring a message flow at deployment time with user-defined properties
Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

Before you begin

For an overview of user-defined properties, see “User-defined properties” on page 569.

For an example of how to code a UDP statement, see DECLARE statement.

About this task

In ESQL, you can define UDPs at the module or schema level. After a UDP has been defined in the Message Flow editor, you can modify the value before you deploy it. You can also modify the value of a UDP dynamically at run time by using the administration REST API, as described in “Setting message flow user-defined properties at run time by using the administration REST API” on page 436.

To configure UDPs, complete the following steps.

Procedure

1. Open the BAR file.

   The contents of the BAR file are shown on the **Manage** tab of the BAR File editor. On this tab, you can expand a flow to show the individual nodes that it contains.
2. Click the relevant message flow or subflow (not the `.cmf` compiled message flow file).

   The UDPs that are defined in that flow are displayed with their values in the **Properties** view.
3. If the value of the UDP is unsuitable for your current environment or task, change it to an appropriate value.

   The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow.

   Note: If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
4. Save your BAR file.

What to do next

Deploy the message flow by following the instructions in "Deploying integration solutions to a production environment" on page 2480.

Accessing a user-defined policy by using ESQL
Use ESQL to query properties dynamically at run time, in user-defined policies.

Before you begin

Complete the following tasks:

- "Creating a message flow" on page 574
- User Defined policy (UserDefined)

About this task

If you created a user-defined policy, and created properties for that policy, you can query those properties by using a Java class method that can be called from ESQL. For example, you can create a user-defined policy to set timeouts for processing HTTP messages.

Procedure

1. Create a Java class with a static method that can be called from the ESQL `mypolicyaccessclass.getpolicyproperty()`.

   In this method, you can access the policy by using the provided Java class `mbpolicy`, as shown in "Accessing a user-defined policy from a JavaCompute node" on page 1785.
2. Right-click the Compute node and click **Open ESQL** to create and open an ESQL file in the Editor view, or open an existing file.
3. Create a Java class called `MyPolicyAccessClass` with a static method called `getPolicyProperty()`.

   This method uses the MbPolicy Java class to return a Java string that contains the policy property. For more information, see "Policies overview" on page 324.
4. Add user code as shown below. In the example shown, a user-defined policy called MyUDCS has been defined in a policy project called PP1.

```
CREATE FUNCTION getPolicyProperty( IN policyName CHARACTER, IN propertyName CHARACTER )
  RETURNS CHARACTER
  LANGUAGE JAVA
  EXTERNAL NAME "MyPolicyAccessClass.getPolicyProperty";

DECLARE Property2 CHARACTER;
```

```
SET Property2=getPolicyProperty('{PP1}:MyUDCS', 'prop2');
```

This lookup syntax can be qualified by policy project. If no policy project is qualified, the policy project that is defined on the *defaultPolicyProject* property in the `node.conf.yaml` configuration file is used. Null is returned if either the property or the policy does not exist.

5. Create and deploy the Java class that is called from the ESQL code that you added in step 2, to access and return the policy property.
For example:

```
import com.ibm.broker.plugin.MbPolicy;
public class MyPolicyAccessClass {
  public static String getPolicyProperty(String policyName, String propertyName ) {
    String resultPropertyValue = null;
    try {
      MbPolicy myPol = MbPolicy.getPolicy("UserDefined", policyName);
      if (myPol != null) {
        resultPropertyValue = myPol.getPropertyValueAsString(propertyName);
        if (resultPropertyValue == null) {
          System.out.println("Unable to find property '"+propertyName+"' in UserDefined
 policy with name '"+policyName+"'");
        } else {
          System.out.println("Found property '"+propertyName+"' with value
 '"+resultPropertyValue+"' in UserDefined policy with name '"+policyName+"'");
        }
      } else {
        System.out.println("Unable to find UserDefined policy with name '"+policyName+"'");
      }
    } catch (MbException mbe) {
      System.out.println("Exception caught trying to find UserDefined policy with name
 '"+policyName+"'. Exception details: '"+mbe.toString()+"'");
    }
    return resultPropertyValue;
} }
```

Developing Java

When you use the JavaCompute node, you customize it to determine the exact processing that it provides.

About this task

You can add any valid Java code to a JavaCompute node, making full use of the Java user-defined node API to process an incoming message. You can use the Java editing facilities of the Eclipse platform to develop your Java code. These facilities include:

- Code completion
- Integrated Javadoc documentation
- Automatic compilation

The Java user-defined node API includes some extra methods that simplify tasks that involve message routing and transformation. These tasks include accessing named elements in a message tree, setting their values, and creating elements, without the need to navigate the tree explicitly.

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, and you can step through the Java class code for the node.

Developers can use any of the following programming styles:

- Java Plugin API
- Java Architecture for XML Binding (JAXB)
- Document Object Model (DOM)

This section provides the following information on developing Java:

- “Java overview” on page 1752
- “Writing Java” on page 1755

Java overview

The Java code that you provide to modify or customize the behavior of a JavaCompute node is stored in a Java project. IBM App Connect Enterprise uses the Eclipse Java perspective for developing and administering Java files.

The topics in this section introduce the following Java concepts:

- Java isolation
- “Java shared classloader” on page 1752
- “JavaCompute node class loading” on page 1754
- “JavaCompute node class loading by using a policy” on page 1754

Java shared classloader

The Java shared classloader loads all the JAR files that are located within the shared-classes directories. For integration servers that are associated with an integration node, the precedence order of loading is dictated by the directories in which the JAR files are located. For independent integration servers, the JAR files are placed in the shared-classes directory under the integration server work directory for the integration server.

The way in which the JAR files are loaded depends on whether the integration server is associated with an integration node or is independent of an integration node:

- **Integration servers associated with an integration node:**

  JAR files are loaded in the following precedence order:

  1. JAR files placed in the integration server `shared-classes` directory allow only a single defined integration server to access them. Files placed in here are loaded first. No other integration servers can use them.

     Add the JAR files to the following directory:

     – For Windows

       workpath\config\<*my_int_node_name*>\<*my_int_server_label*>\shared-classes

     – For Linux

       workpath/config/<*my_int_node_name*>/<*my_int_server_label*>/shared-classes

     Ensure that the integration node name, and any integration servers created, contain only characters that are valid on your file system. You may also need to create the required directory structure.

     All files placed into the integration server `shared-classes` directory that have a `.jar` extension, are loaded and made available in the Java environment for that integration server. JAR files in this directory take precedence over JAR files in the integration node `shared-classes` directory.

     Note: If the integration server `shared-classes` directory is empty when the integration server is deleted, the directory is automatically removed.

  2. JAR files placed in the integration node `shared-classes` directory allow only a single defined integration node to access them. Files placed in here are loaded after any files placed in the integration server `shared-classes` directory. No other integration node can use them.

     Add the JAR files to the following directory:

     – For Windows

       workpath\config\<*my_int_node_name*>\shared-classes

     – For Linux

       workpath/config/<*my_int_node_name*>/shared-classes

       Ensure that the integration node name contains only characters that are valid on your file system. You may also need to create the required directory structure.

       All files placed into the integration node `shared-classes` directory that have a `.jar` extension, are loaded and made available in the Java environment for all integration servers in that integration node. JAR files in this directory take precedence over JAR files in the top level `shared-classes` directory.

       Note: If the integration node `shared-classes` directory is empty when the integration node is deleted, the directory is automatically removed.

  3. JAR files placed in the top level `shared-classes` directory are made available to all integration node and all integration servers. Files placed in here are loaded after any files placed in the integration node `shared-classes` directory.

     Add the JAR files to the following directory:

     – For Windows

       workpath\shared-classes

     – For Linux

       workpath/shared-classes

- **Independent integration servers:**

  The Java shared classloader loads all the JAR files that are located within the shared-classes directories. JAR files are placed in the shared-classes directory under the integration server work directory for the integration server. For example, if the work directory passed to the command on

startup is `/Users/user1/workDir1`, the shared-classes directory is `/Users/user1/workDir1/shared-classes`.

JavaCompute node class loading
The JavaCompute node loads and runs a Java class that is defined as the `Java class` property on the node. This class is deployed, with any other required classes, in a Java archive (JAR) file.

You can deploy this JAR file in several ways; for example:

- You can deploy the JAR file in a BAR file with the container of the message flow that contains the JavaCompute node.
- You can deploy the JAR file directly to an integration server in a shared library that is referenced by one or more applications or shared libraries.
- You can deploy the JAR file in a BAR file in a shared library that is referenced by one or more applications or shared libraries.

Java classes in shared libraries

You can use a shared library to share the Java classes between multiple solutions, or to provide isolation.

Java classes that are deployed in a shared library are not available to the integration server-wide class loader. When you deploy a shared library that contains Java files, a new class loader is created for that shared library. This class loader contains all Java classes in the shared library, as well as the Java classes from all referenced shared libraries. Delegation does not exist from one shared library class loader to another shared library class loader. Java classes in shared library class loaders are isolated from Java classes in the integration server-wide class loader. Similarly, Java classes in the integration server-wide class loader are isolated from the Java classes in any shared library class loaders.

A Java class can exist in multiple class loaders if that Java class is contained in a shared library that is referenced by other shared libraries. When you update that Java class by redeploying the shared library that contains it, all class loaders for all shared libraries that contain that Java class are deleted and re-created.

Integration server-wide class loader

Java classes that are deployed in a BAR file in the container of the flow that includes the JavaCompute node are loaded by an integration server-wide class loader. Whenever a new or changed JAR file is deployed, the integration server-wide class loader is deleted and re-created with all the currently deployed JAR files. At the same time, all JavaCompute nodes refresh the Java classes that are being used within them, as well as re-creating all the Java static variables. You can modify this behavior by using the `Java classloader service` property on the node, which allows alternative class loaders to be used. For more information, see "JavaCompute node class loading by using a policy" on page 1754.

The integration server-wide class loader first searches all the deployed JAR files for a required class. If a required class cannot be found, it defers to the shared class loader. The shared class loader looks in a set of directories on the integration node machine and loads any JAR files found. It can be used to install any required JAR files that do not need to be repeatedly deployed, such as client libraries that the JavaCompute nodes need to use.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files that are installed in the shared classes directories, a class loader that contains all the integration node supplied classes is checked (for example: this class loader contains the `jplugin2.jar`), followed by the class path, and then finally the Java virtual machine (JVM) system class loader.

Delegation from one class loader to another can occur in one direction only. If a class is resolved in the shared class loader, it cannot create classes directly in the integration server-wide class loader.

JavaCompute node class loading by using a policy
The JavaCompute node loads and runs a Java class that is defined as the `Java class` property on the node. Normally, this class is deployed, along with any other required classes, in a Java archive (JAR) file that is contained in a BAR file. Any Java classes that are deployed in this way are loaded by an integration

server-wide class loader. You can override this behavior by setting the `Javaclass loader service` property on the node to the name of a Java Class Loader policy.

As an alternative to using a policy you can use shared libraries to provide isolation for deployed Java classes. Java classes that are deployed in a shared library are not available to the integration server-wide class loader. When you deploy a shared library that contains Java files, a new class loader is created for that shared library (see "JavaCompute node class loading" on page 1754).

The class loader that is defined by the Java Class Loader policy has a list of JAR files (defined by the `JAR files to be loaded` property), which it owns and uses. Whenever these JAR files are deployed, the policy class loader receives the JAR files and uses them for resolving classes. The JAR files are no longer available to the integration server class loader, and any node that uses that class loader does not have access to any classes that are contained in those JAR files.

The policy class loader first searches all the deployed JAR files that it has received for a required class. If a required class cannot be found, the policy class loader defers to the shared class loader. The shared class loader looks in a set of directories on the integration server machine and loads any JAR files that are found. The shared class loader can be used to install required JAR files that do not need to be deployed repeatedly, such as client libraries that the JavaCompute nodes need to use. You can override this mechanism by setting the `Shared JAR file path` property on the policy to use a specified directory to find installed JAR files, rather than the shared classes directories.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files that are installed in the shared classes directories, a class loader that contains all the integration server-supplied classes is checked (for example, this class loader contains the `jplugin2.jar`), followed by the classpath, and then the Java virtual machine (JVM) system class loader.

Properties of the Java Class Loader policy are defined in Java Class Loader policy (JavaClassLoader).

Writing Java

When you create a message flow, you include input nodes that receive messages and, optionally, output nodes that send out new or updated messages. If the message processing requires it, you can include other nodes after the input node that are customized in Java to complete the actions that your applications need.

About this task

You can customize the processing that some of the built-in nodes provide. In a JavaCompute node, you can provide Java code that controls precisely the behavior of the node. This set of topics discusses how you can use Java to customize the JavaCompute node.

You can use a JavaCompute node to check and manipulate message content. The node can read the contents of the input message, then construct new output messages that are created from all, part, or none of the input message.

You can also use a JavaCompute node to interact with a global cache. You can write Java code that stores data in a global cache, or retrieves data for further processing or routing.

Use the Debug perspective to debug a message flow that contains a JavaCompute node. When control passes to a JavaCompute node during debugging, the perspective opens the Java debugger, allowing you to step through the Java class code for the node.

This section provides more information about writing Java:

- "Creating Java code for a JavaCompute node" on page 1756
- "Opening an existing Java file" on page 1758
- "Saving a Java file" on page 1759
- "Adding Java code dependencies" on page 1759
- "Deploying JavaCompute node code" on page 1760
- Manipulating message body data
- Manipulating other parts of the message tree

- "Accessing the global cache by using a JavaCompute node" on page 1781
- Accessing user-defined properties
- "Accessing a user-defined policy from a JavaCompute node" on page 1785
- "Executing ODM business rules by using a JavaCompute node" on page 1245
- "Creating ODM classes by using the Process via Operational Decision Manager ruleset wizard" on page 1247
- Interacting with databases
- "Calling an Enterprise Java Bean" on page 1787
- Handling exceptions
- Logging errors

Creating Java code for a JavaCompute node
You can associate Java code with your JavaCompute node by using a wizard, setting node properties, or creating a Java project.

Before you begin

Before you complete this task, create a JavaCompute node in your message flow.

About this task

You can store the Java code for a JavaCompute node in the same application as the message flow that contains the JavaCompute node. Alternatively, if you want to share Java classes across multiple solutions, you can store Java code in a shared library and refer to that library from one or more applications that contain message flows with JavaCompute nodes. In this case, if you already have Java classes, you can add them to a shared library before you create your message flows. Alternatively, you can specify a shared library to contain the classes when you create the JavaCompute node.

To associate Java code with a JavaCompute node, use one of the following methods.

- "Use the New Java Compute Node Class wizard to create template code" on page 1756
- "Associate a JavaCompute node with an existing Java class" on page 1757
- "Create a Java project from scratch" on page 1757

*Use the **New Java Compute Node Class** wizard to create template code*

About this task

Use the **New Java Compute Node Class** wizard to create template code.

Procedure

1. In the IBM App Connect Enterprise Toolkit, right-click the JavaCompute node and click **Open Java**.
2. Follow the instructions in the **New Java Compute Node Class** wizard.
   a) On the **Java Compute Node Class** page, set **Source folder** to the application or library that contains the Java classes. Click **Next**.
   b) On the **Java Compute Node Class Template** page, choose one of the following options:
      - For a filter node template, select **Filtering message class**.
      - To change an incoming message, select **Modifying message class**.
      - To create a new message, select **Creating message class**.
      - To process messages by using the JAXB template, select **Process via JAXB class**. If you want to use the **Generate JAXB Java object classes** wizard to create your JAXB Java object classes, or

to update the template code to reference your existing JAXB Java object classes, click **Next**. For more information, see "Creating JAXB Java object classes by using a wizard" on page 1774.

- To execute business rules, select **Process via Operational Decision Manager ruleset**. If you want to use the wizard, click **Next**.

c) Optional: If necessary, change the default Java build settings. Click **Next**.

d) Optional: If necessary, change the default JavaCompute node project name.

3. Click **Finish**.

Results

The Java class file opens in the editor. If you view the basic properties for the JavaCompute node, the **Java class** field is populated with the location of the specified Java class. If your Java classes are stored in a shared library, the **Java classloader service** field contains a qualifier for the shared library that contains the Java class in the form {*sharedLibraryName*}.

Associate a JavaCompute node with an existing Java class

About this task

You can associate a JavaCompute node with an existing Java class that the wizard has generated previously. You can use this method to share the same Java code between multiple nodes.

To associate a JavaCompute node with an existing Java class, complete the following steps:

Procedure

1. In the IBM App Connect Enterprise Toolkit, right-click the JavaCompute node and click **Properties**.
2. In the **Java Class** field, click **Browse** and start to type the name of the required Java class.
   As you type, matching classes are displayed.
3. Select the appropriate class and click **OK**.

Results

The **Java class** field is populated with the location of the specified Java class. If your Java classes are stored in a shared library, the **Java classloader service** field contains a qualifier for the shared library that contains the Java class in the form {*sharedLibraryName*}. View the Java class by double-clicking the JavaCompute node.

Create a Java project from scratch

About this task

You can create a Java project from scratch. Before you add one or more classes to the project, complete the following steps.

Procedure

1. Open the `.project` file in the text editor, and ensure that the following builders and natures are set:

```
<buildSpec>
  <buildCommand>
   <name>org.eclipse.jdt.core.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.java.builder.javabuilder</name>
   <arguments>
   </arguments>
  </buildCommand>
  <buildCommand>
   <name>com.ibm.etools.mft.jcn.jcnbuilder</name>
```

```
    <arguments>
    </arguments>
   </buildCommand>
   <buildCommand>
    <name>com.ibm.etools.mft.bar.barbuilder</name>
    <arguments>
    </arguments>
   </buildCommand>
  </buildSpec>
  <natures>
   <nature>org.eclipse.jdt.core.javanature</nature>
   <nature>com.ibm.etools.mft.bar.barnature</nature>
   <nature>com.ibm.etools.mft.jcn.jcnnature</nature>
  </natures>
```

2. Add the following plug-ins to the build path of the Java project:
   a. Open the properties of the Java project.
   b. Select **Java Build Path** and open the **Libraries** tab.
   c. Click **Add Variable**.
   d. Select the variable `JCN_HOME` and click **OK**.
   e. Double-click the variable you added to open the **Edit Variable Entry** dialog.
   f. Click **Extension** and select `javacompute.jar`.
   g. Repeat the previous four steps to add the variable `JCN_HOME/jplugin2.jar`.
3. Create the appropriate Java class and ensure that it extends from `com.ibm.broker.javacompute.MbJavaComputeNode`.

Results

You have created your Java project.

What to do next

You can now complete the following tasks:

- “Opening an existing Java file” on page 1758
- “Saving a Java file” on page 1759
- “Adding Java code dependencies” on page 1759

Opening an existing Java file
You can add to and modify Java code that you have created in a Java project.

Before you begin

Before you start this task, complete the following tasks:

- Add a node to your message flow.
- “Creating Java code for a JavaCompute node” on page 1756

About this task

To open an existing Java file:

Procedure

1. Switch to the Java perspective.
2. In the Package Explorer view, double-click the Java file that you want to open.
   The file is opened in the editor view.
3. Work with the contents of the file to make your changes.

Results

You can also open a Java file when you have a message flow open in the editor view. Right-click the JavaCompute node, then select **Open Java**.

What to do next

You can now perform the following tasks:

- "Saving a Java file" on page 1759
- "Adding Java code dependencies" on page 1759

Saving a Java file

When you edit your Java files, save them to preserve the additions and modifications that you have made.

Before you begin

To complete this task, you must have completed the following tasks:

- Add a node to your message flow.
- "Creating Java code for a JavaCompute node" on page 1756

About this task

To save a Java file:

Procedure

1. Switch to the Java perspective.
2. Create a new Java file or open an existing Java file.
3. Make the changes to the contents of the Java file.
4. When you have finished working, click **File** > **Save** or **File** > **Save All** to save the file and retain all your changes.

What to do next

You can now perform the following task:

- "Adding Java code dependencies" on page 1759

Adding Java code dependencies

When you write Java code for a JavaCompute node, you can include references to other Java projects and JAR files.

Before you begin

To complete this task, you must have completed the following tasks:

- Add a node to your message flow.
- Create Java code for a JavaCompute node.

About this task

The Java code in a JavaCompute node might contain references to other Java projects in the Eclipse workspace (internal dependencies), or to external JAR files, for example the JavaMail API (external dependencies), or a set of JAXB Java object classes (internal or external). If other JAR files are referenced, you must add the files to the project class path.

See "Using JAXB with a JavaCompute node" on page 1772 for an example of a project using Java objects generated by the JAXB binding compiler.

Procedure

1. Right-click the project folder for the project on which you are working, and click **Properties**.
2. Click **Java Build Path** in the left pane.
3. Click the **Libraries** tab.
4. Complete one of the following steps:
   - To add an internal dependency, click **Add JARs**, select the JAR file that you want to add, then click **OK**.
   - To add an external dependency, click **Add External JARs**, select the JAR file that you want to add, then click **Open**. Copy the JAR file to the `shared-classes` directory required. If you do not copy the JAR file to a valid `shared-classes` directory, `ClassNotFoundException` exceptions are generated at run time.

Results

You have now added a code dependency.

Deploying JavaCompute node code
You can deploy Java code for a JavaCompute node in a BAR file or independently in another container, such as a shared library.

Before you begin

Complete the following tasks:

1. Create a message flow that contains a JavaCompute node.
2. Create the Java code for the JavaCompute node.

About this task

After you create a JavaCompute node and the Java code that it uses, you can deploy your resources. The method of deployment depends on where you store your resources.

You can store the Java code for a JavaCompute node in the same application as the message flow that contains the JavaCompute node. Alternatively, to share Java classes across multiple solutions, store Java code in a shared library and refer to that library from one or more applications that contain JavaCompute nodes.

Use one of the following methods to deploy your Java code.

Procedure

- If your Java code is in the same application as the JavaCompute node, deploy the application by adding it to a BAR file or by dragging it to an integration server. When you deploy an application that contains a JavaCompute node, the IBM App Connect Enterprise Toolkit packages the compiled Java code and its dependencies into the BAR file.
- If your Java code is contained in a referenced shared library, deploy the shared library by dragging it onto the integration server or adding it to a BAR file. You must deploy the shared library before or with the applications that reference it.

Results

Your resources are deployed to the integration server. You can view them in the Integration nodes view.

When you update and redeploy the Java code in a shared library, those updates are available automatically to all the applications that reference that shared library. In this case, you do not need to redeploy those applications.

Manipulating message body data by using a JavaCompute node
You can manipulate message body data by using a JavaCompute node.

About this task

The message body is always the last child of root, and its parser name identifies it, for example XML or MRM.

The following topics describe how to refer to, modify, and create message body data. The information provided here is domain independent:

- "Accessing elements in a message tree from a JavaCompute node" on page 1761
- "Transforming a message by using a JavaCompute node" on page 1764
- "Creating a simple filter by using a JavaCompute node" on page 1769
- "Propagating a message to the JavaCompute node Out and Alternate terminals" on page 1770
- "Extracting information from a message by using XPath 1.0 and a JavaCompute node" on page 1770

Accessing elements in a message tree from a JavaCompute node
Access the contents of a message, for reading or writing, using the structure and arrangement of the elements in the tree that the parser creates from the input bit stream.

About this task

Follow the relevant parent and child relationships from the top of the tree downwards until you reach the required element.

The message tree is passed to a JavaCompute node as an argument of the `evaluate` method. The argument is an MbMessageAssembly object. MbMessageAssembly contains four message objects:

- Message
- Local Environment
- Global Environment
- Exception List

These objects are read-only, except for Global Environment. If you try to write to the read-only objects, an MbReadOnlyException is issued.

Alternatively, use JAXB Java objects with getter and setter methods; see "Using JAXB with a JavaCompute node" on page 1772.

This topic contains the following information about accessing elements in a message tree:

- "Traversing the element tree" on page 1761
- "Accessing information about an element" on page 1763

Traversing the element tree

About this task

The following table shows the Java methods that you can use to access element trees, and the equivalent ESQL field type constant for each point in the tree.

Java accessor from MbMessageAssembly	ESQL field type constant
getMessage().getRootElement()	InputRoot
getMessage().getRootElement().getLastChild()	InputBody
getLocalEnvironment().getRootElement()	InputLocalEnvironment
getGlobalEnvironment().getRootElement()	Environment

Java accessor from MbMessageAssembly	ESQL field type constant
getExceptionList().getRootElement()	InputExceptionList

Use the following methods to traverse a message tree from an element of type MbElement:

getParent()
returns the parent of the current element

getPreviousSibling()
returns the previous sibling of the current element

getNextSibling()
returns the next sibling of the current element

getFirstChild()
returns the first child of the current element

getLastChild()
returns the last child of the current element

Alternatively, use a Document Object Model (DOM) object, and use the DOM API to navigate and manipulate the elements.

MBMessage provides the following methods:

getDOMDocument()
obtains a W3C `org.w3c.dom.Document` object for the message body.

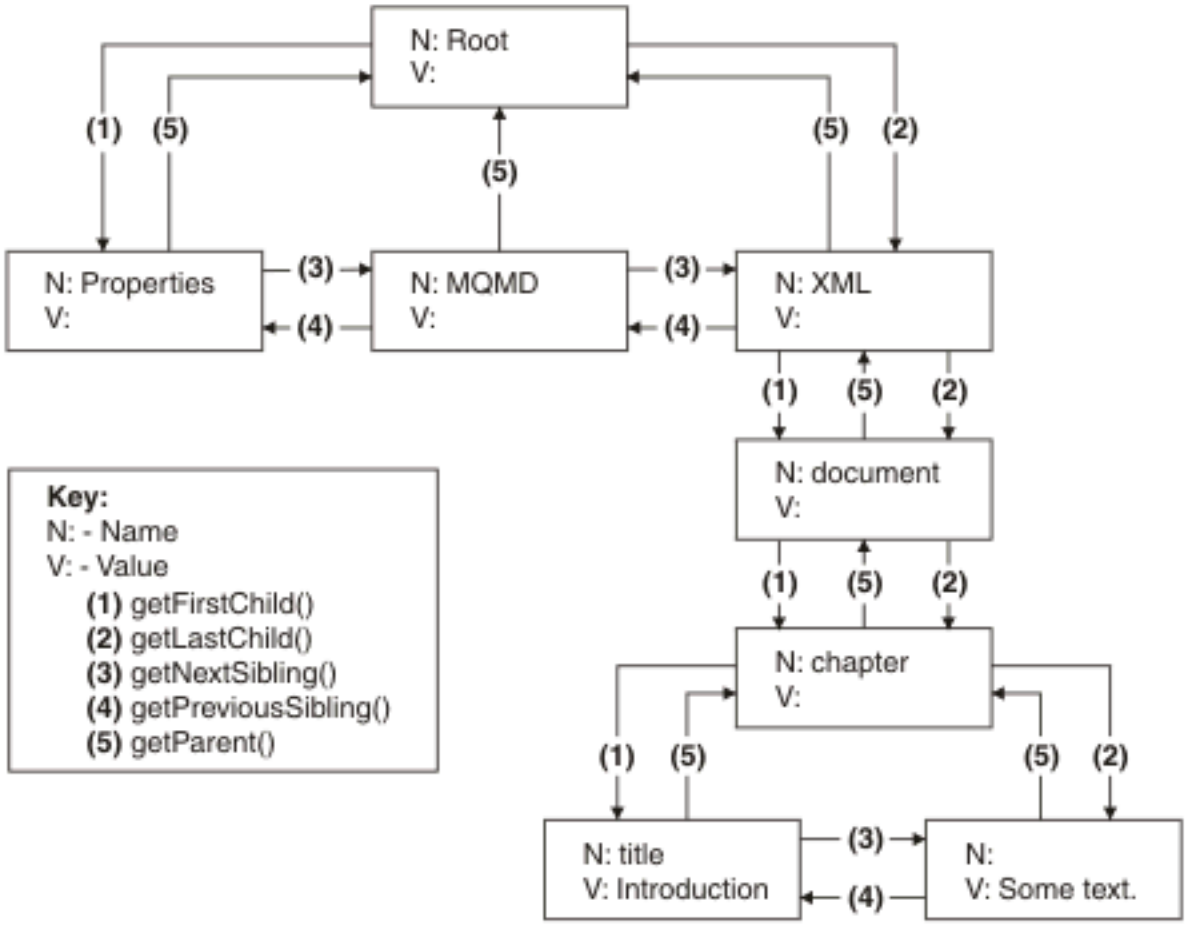
createDOMDocument()
obtains a W3C `org.w3c.dom.Document` object for the message body.

MbElement provides the following method:

getDOMNode()
obtains a W3C `org.w3c.dom.Node` object for the message element.

The following example shows a simple XML message and the logical tree that would be created from the message. The message has been sent using IBM MQ. The logical tree diagram also shows the methods to call in order to navigate around the tree.

```
<document>
  <chapter title='Introduction'>
    Some text
  </chapter>
</document>
```



The tree used in this diagram is the one that is created by parsing the previous XML example.

- From the Root part of the tree, calling **`getFirstChild()`** navigates to Properties. Also from Root, calling **`getLastChild()`** returns XML.
- From Properties, calling **`getParent()`** returns Root, and calling **`getNextSibling()`** returns MQMD.
- From MQMD, calling **`getPreviousSibling()`** returns Properties, calling **`getParent()`** returns Root, and calling **`getNextSibling()`** returns XML.
- From XML, calling **`getPreviousSibling()`** returns MQMD, calling **`getParent()`** returns Root, calling **`getFirstChild()`** returns document, and calling **`getLastChild()`** also returns document.
- From document, calling **`getParent()`** returns XML, calling **`getFirstChild()`** returns chapter, and calling **`getLastChild()`** also returns chapter.
- From chapter, calling **`getParent()`** returns document, calling **`getFirstChild()`** returns title, and calling **`getLastChild()`** returns the child that contains the message data "Some text.".

The following Java code accesses the chapter element in the logical tree for an XML message that does not contain white spaces. The XML parser retains white space in the parsed tree, but the XMLNS and XMLNSC parsers do not.

```
MbElement root = assembly.getMessage().getRootElement();
MbElement chapter = root.getLastChild().getFirstChild().getFirstChild();
```

Accessing information about an element

About this task

Use the following methods to return information about the referenced element:

getName()
returns the element name as a java.lang.String

getValue()
returns the element value

getType()
returns the generic type, which is one of the following types:

- NAME: an element of this type has a name, but no value.
- VALUE: an element of this type has a value, but no name.
- NAME/VALUE: an element of this type has both a value and a name.

getSpecificType()
returns the parser-specific type of the element

getNamespace()
returns the namespace URI of this element

Transforming a message by using a JavaCompute node
You can transform a message by using the JavaCompute node.

The following topics describe how to transform messages by using the JavaCompute node:

- “Creating a new message by using a JavaCompute node” on page 1764
- “Copying and modifying a message by using a JavaCompute node” on page 1765
- “Setting, copying, and moving message elements by using a JavaCompute node” on page 1765
- “Creating new elements by using a JavaCompute node” on page 1766
- “Using JAXB with a JavaCompute node” on page 1772
- “Processing message body data by using JAXB Java object classes” on page 1775
- “Working with large input messages to propagate multiple output messages” on page 1767

Creating a new message by using a JavaCompute node
Many message transformation scenarios require a new outgoing message to be built. The *Create Message Class* template in the **JavaCompute node** wizard generates template code for this.

About this task

In the template code, the default constructor of MbMessage is called to create a blank message, as shown in the following Java code:

```
MbMessage outMessage = new MbMessage();
```

The headers can be copied from the incoming message by using the supplied utility method, copyMessageHeaders(), as shown in this Java code:

```
copyMessageHeaders(inMessage, outMessage);
```

The new message body can now be created. First, add the top level parser element. For XML, this is:

```
MbElement outRoot = outMessage.getRootElement();
MbElement outBody = outRoot.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
```

The remainder of the message can then be built up by using the **createElement** methods and the extended syntax of the integration node XPath implementation.

When you want to create a BLOB message, that is handled as a single byte string by using the BLOB parser domain. The message data is added as a byte array to the single element named "BLOB" under the parser level element, described as follows:

```
String myMsg = "The Message Data";
MbElement outRoot = outMessage.getRootElement();
```

```
// Create the Integration Bus Blob Parser element
MbElement outParser = outRoot.createElementAsLastChild(MbBLOB.PARSER_NAME);
// Create the BLOB element in the Blob parser domain with the required text
MbElement outBody = outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, "BLOB",
 myMsg.getBytes());
```

You can use the following code to create a BLOB message in a JavaCompute node:

```
MbElement outMsgRootEl = outMessage.getRootElement();

String parserName = MbBLOB.PARSER_NAME;
String messageType = "";
String messageSet = "";
String messageFormat = "";
int encoding = 0;
int ccsid = 0;
int options = 0;
outMsgRootEl.createElementAsLastChildFromBitstream(responseBodyXmlData,
  parserName, messageType, messageSet, messageFormat, encoding, ccsid,
  options);
```

To add a JSON message in a JavaCompute node, see “Creating a JSON message” on page 1745.

Copying and modifying a message by using a JavaCompute node
You can use the *Modify Message Class* template in the JavaCompute node wizard to generate a copied or modified version of an input message

About this task

The incoming message and message assembly are read-only. To modify a message, a copy of the incoming message must be made. The *Modifying Message Class* template in the JavaCompute node wizard generates this copy. In the template code, the copy constructor of *MBMessage* is called to create a message that is populated by copying the input, as shown in the following Java code:

```
MbMessage outMessage = new MbMessage(inAssembly.getMessage());
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
```

The new outAssembly, outMessage objects, or both, can now be modified and propagated to the next node.

Setting, copying, and moving message elements by using a JavaCompute node
Transform elements in the message as it passes through a JavaCompute node in the message flow.

Before you begin

The incoming message and message assembly are read-only. Therefore, to modify a message, you must make a copy of it first. For more information, see “Copying and modifying a message by using a JavaCompute node” on page 1765.

About this task

- “Setting information about an element” on page 1765
- “Moving and copying elements” on page 1766

The Java API reference information provides details about each of the methods used in the following sections:

Setting information about an element

About this task

Use these methods to set information about the referenced element:

setName()
Sets the name of the element

setValue()
Sets the value of the element

setSpecificType()
Sets the parser-specific type of the element

setNamespace()
Sets the namespace URI of the element

Moving and copying elements

About this task

Use a JavaCompute node to copy or detach an element from a message tree by using the following methods:

`detach()`
The element is detached from its parent and siblings, but any child elements are left attached

`copy()`
A copy of the element and its attached children is created

Use one of four methods to attach an element or subtree that you have copied on to another tree:

`addAsFirstChild(`*element*`)`
Adds an unattached element as the first child of *element*

`addAsLastChild(`*element*`)`
Adds an unattached element as the last child of *element*

`addBefore(`*element*`)`
Adds an unattached element as the previous sibling of *element*

`addAfter(`*element*`)`
Adds an unattached element as the next sibling of *element*

Creating new elements by using a JavaCompute node
You can use a JavaCompute node to create new elements.

About this task

Use the following methods in a JavaCompute node to create new elements in a message tree:

- createElementAsFirstChild()
- createElementAsLastChild()
- createElementBefore()
- createElementAfter()

The method returns a reference to the newly-created element. Each method has three overloaded forms:

createElement...(int type)
Creates a blank element of the specified type. Valid generic types are:

- MbElement.TYPE_NAME. This type of element has only a name, for example an XML element.
- MbElement.TYPE_VALUE. This type of element has only a value, for example XML text that is not contained in an XML element.
- MbElement.TYPE_NAME_VALUE. This type of element has both a name and a value, for example an XML attribute.

Specific type values can also be assigned. The meaning of this type information is dependent on the parser. Element name and value information must be assigned by using the setName() and setValue() methods.

createElement...(int type, String name, Object value)
Method for setting the name and value of the element at creation time.

createElement...(String parserName)
A special form of createElement...() that is only used to create top-level parser elements.

This example Java code adds a new chapter element to the XML example given in "Accessing elements in a message tree from a JavaCompute node" on page 1761:

```
MbElement root = outMessage.getRootElement();
MbElement document = root.getLastChild().getFirstChild();
MbElement chapter2 = document.createElementAsLastChild(MbElement.TYPE_NAME,"Chapter",null);

// add title attribute
MbElement title2 = chapter2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
 "title", "Message Flows");
```

This produces the following XML output:

```
<document>
 <chapter title="Introduction">
  Some text.
 </chapter>
 <chapter title="Message Flows"/>
</document>
```

Working with large input messages to propagate multiple output messages
The tree representation of a message is typically bigger than the input bit stream. Manipulating a large message tree can require much storage but you can code Java methods that help to reduce the storage load on the integration node.

About this task

When an input bit stream is parsed and a logical tree is created, the tree representation of a message is typically bigger, and in some cases much bigger, than the corresponding bit stream.

The reasons for this expansion include the following factors:

- The addition of the pointers that link the objects together
- Translation of character data into Unicode, which can double the size
- The inclusion of field names that might have been implicit in the bit stream
- The presence of control data that is associated with operation of the integration node

Manipulating a large message tree can require much storage. If you design a message flow that handles large messages that are made up of repeating structures, you can code Java methods that help to reduce the storage load on the integration node. These methods support both random and sequential access to the message, but assume that you do not need access to the whole message at one time.

These Java methods cause the integration node to complete limited parsing of the message, and to keep in storage at one time, only that part of the message tree that reflects a single record. If your processing requires you to retain information from record to record (for example, to calculate a total price from a repeating structure of items in an order), you can either declare, initialize, and maintain Java variables, or you can save values in another part of the message tree; for example, in the local environment.

This technique reduces the memory that is used by the integration node to the memory that is needed to hold the full input and output bit streams, plus the memory that is needed for the message trees of just one record. This technique also provides memory savings when even a few repeats are encountered in the message. The integration node uses partial parsing and the ability to parse specified parts of the message tree, to and from the corresponding part of the bit stream.

To use these techniques in your JavaCompute node (or Java user-defined node), take any of the following steps.

- Copy the input MbMessage by using the copy constructor. This action creates a modifiable copy of the input message that is not parsed and therefore uses a minimum amount of memory.
- Avoid any inspection of the input message, which avoids the need to parse the message.

- Use a loop and an MbElement variable to step through the message one record at a time. For each record, use the following processes:
  - Use normal transforms to build a corresponding output subtree in an output MbMessage.
  - Use the MbElement.delete() method to delete the current input record message tree when you have completed the record manipulation.
  - Use the following code to propagate the output message for the current record.

    ```
    MbOutputTerminal.propagate(MbMessageAssembly, true)
    ```

    This process allows message tree resources and parsers to be recovered and reused for the next output message iteration.

You can vary these techniques to suit the processing that is required for your messages.

The following example Java code demonstrates how to parse a large input message with many repeating records, where each record is propagated as an individual output message.

The input XMLNSC messages are in the following form, where there are many repeating `Record` elements.

```
<TestCase>
  <Record><Field1>A</Field1><Field2>B</Field2><Field3>C</Field3><Field4>D</Field4><Field5>EA</Field5></
Record>
  <Record><Field1>A</Field1><Field2>B</Field2><Field3>C</Field3><Field4>D</Field4><Field5>EA</Field5></
Record>
  <Record><Field1>A</Field1><Field2>B</Field2><Field3>C</Field3><Field4>D</Field4><Field5>EA</Field5></
Record>
  <Record><Field1>A</Field1><Field2>B</Field2><Field3>C</Field3><Field4>D</Field4><Field5>EA</Field5></
Record>
  ....
</TestCase>
```

The following example shows the Java code that can be used to parse this large message, one record at a time, and propagate each record.

```
//Make a modifiable MbMessage based on the input message passed in on the inAssembly.
MbMessage clonedInMessage = new MbMessage(inAssembly.getMessage());
//Now partially parse the cloned input message one record at a time.
MbElement inputRootElement = clonedInMessage.getRootElement();
MbElement inputPropertiesElement = inputRootElement.getFirstElementByPath("Properties");
MbElement inputMQMDElement = inputRootElement.getFirstElementByPath("MQMD");
MbElement inputXMLNSCElement = inputRootElement.getFirstElementByPath("XMLNSC");

MbElement inputXMLNSCRootTagElement = inputXMLNSCElement.getFirstChild(); //Move to the TestCase tag
MbElement currentInputRecord = inputXMLNSCRootTagElement.getFirstChild(); //Move to the Record tag

while(currentInputRecord != null)
{
  // Create a new output message for the record that we are going to be propagate.
  MbMessage outputMessage = new MbMessage();
  MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outputMessage);

  //Create new parsers folders in the output message.
  MbElement outputRootElement = outputMessage.getRootElement();
  MbElement outputPropertiesElement =
 outputRootElement.createElementAsLastChild(inputPropertiesElement.getParserClassName());
  MbElement outputMQMDElement =
 outputRootElement.createElementAsLastChild(inputMQMDElement.getParserClassName());
  MbElement outputXMLNSCElement =
 outputRootElement.createElementAsLastChild(inputXMLNSCElement.getParserClassName());
  //Create the root tag in the output XMLNSC folder that will be used for this output record.
  MbElement outputXMLNSCRootTag = outputXMLNSCElement.createElementAsLastChild(MbElement.TYPE_NAME,
 "TestCase", null);
  //Create the record tag for this output message instance.
  MbElement currentOutputRecord = outputXMLNSCRootTag.createElementAsLastChild(MbElement.TYPE_NAME,
 "Record", null);

  //Copy the Properties Folder, MQMD header, and the current record.
  outputPropertiesElement.copyElementTree(inputPropertiesElement);
  outputMQMDElement.copyElementTree(inputMQMDElement);
  currentOutputRecord.copyElementTree(currentInputRecord);

  //Propagate this message, requesting that the output message assembly be cleared after propagation.
```

```
  out.propagate(outAssembly, true);
  //Note: You do not need to call clearMessage on outputMessage because it was cleared after the
 propagation.

  //Take a reference to the current record so that it can be deleted.
  MbElement previousInputRecord = currentInputRecord;
  //Now move to the next input record ready for processing.
  currentInputRecord = currentInputRecord.getNextSibling();
  //Now that we have moved to the next sibling, delete the input record that has already been processed.
  previousInputRecord.delete();

}
```

This Java code produces a message for each repeating Record subtree in the input XMLNSC message:

```
<Record>
  <Field1>A</Field1>
  <Field2>B</Field2>
  <Field3>C</Field3>
  <Field4>D</Field4>
  <Field5>E</Field5>
</Record>
```

Creating a simple filter by using a JavaCompute node
The JavaCompute node has two output terminals: Out and Alternate. To use the JavaCompute node as a filter node, propagate a message to either the Out or Alternate terminal based on the message content.

Before you begin

To complete this task, you must have added a node to your message flow.

About this task

Use the **JavaCompute node creation** wizard to generate template code for a filter node:

Procedure

Select the *Filtering Message Class* template in the **JavaCompute node creation** wizard to create a filter node.

The following template code is produced. It passes the input message to the Out terminal without doing any processing on the message.

```
public class jcn2 extends MbJavaComputeNode {

  public void evaluate(MbMessageAssembly assembly) throws MbException {
    MbOutputTerminal out = getOutputTerminal("out");
    MbOutputTerminal alt = getOutputTerminal("alternate");

    MbMessage message = assembly.getMessage();

    // ----------------------------------------------------------
    // Add user code below

    // End of user code
    // ----------------------------------------------------------

    // The following should only be changed
    // if not propagating message to the 'out' terminal

    out.propagate(assembly);
  }
}
```

The template produces a partial implementation of a method called `evaluate()`. The integration node calls `evaluate()` once for each message that passes through the node. The parameter that is passed to `evaluate()` is the message assembly. The message assembly encapsulates the message that is passed on from the previous node in the message flow.

Add custom code to the template, and propagate messages to both the Out and Alternate terminals to create a message filter.

Propagating a message to the JavaCompute node Out and Alternate terminals
The JavaCompute node has two output terminals, Out and Alternate. Therefore, you can use the node both as a filter node and as a message transformation node. After you process the message, propagate the message to an output terminal by using a `propagate()` method.

About this task

To propagate the message assembly to the Out terminal use the following method:

```
out.propagate(assembly);
```

To propagate the message assembly to the Alternate terminal, use the following method:

```
alt.propagate(assembly);
```

To propagate the same MbMessage object multiple times, call the `finalizeMessage()` method on the MBMessage object, so that any changes made to the message are reflected in the bit stream that is generated downstream of the JavaCompute node; for example:

```
MbMessage outMessage = new MbMessage(inMessage);
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly, outMessage);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);
```

When you propagate many times from a JavaCompute node, where a new MbMessage is created each time, use the following code:

```
MbOutputTerminal.propagate(MbMessageAssembly, true)
```

This code recovers the message tree and parser resources after the propagation so that these resources can be used when the next MbMessage is constructed for propagation. You can see an example in "Working with large input messages to propagate multiple output messages" on page 1767.

When you use this code, the message tree resources are reclaimed for the non-read-only MbMessages in the MbMessageAssembly. As a result, the method MbMessage.clearMessage(true) is called on each modifiable MbMessage; therefore, these MbMessages cannot be used again.

If message tree fields were local to each MbMessage, or were only detached or attached between the three MbMessages in the MbMessageAssembly, the parsers are also recovered for reuse. However, if elements were detached and attached to an MbMessage that was not propagated, the parsers cannot be reused on the next iterations of input records.

For such MbMessage objects that were not propagated, call the method MbMessage.clearMessage(true) explicitly before the next input record is processed. This method allows parsers to be reused.

Extracting information from a message by using XPath 1.0 and a JavaCompute node
XPath is a query language designed for use with XML documents, but you can use it with any tree structure to query contents.

IBM App Connect Enterprise uses XPath to select elements from the logical message tree regardless of the format of the bit stream. The terminology used in this topic is based on the terminology used in the W3C definition of XPath 1.0. For more information about XPath, see "Using XPath" on page 626; and for more information about the W3C definition of the XPath 1.0 standard, see W3C XPath 1.0 Specification. For examples of XPath use, see the MbXPath topic in the Java user-defined node API documentation.

This topic contains the following information:

- “XPath variable binding” on page 1771
- “XPath namespace support” on page 1771
- “Updating a message by using XPath extensions” on page 1772

Using the evaluateXPath method to extract message information

The `evaluateXPath()` method is included in the Java user-defined node API. It supports XPath 1.0, with the following exceptions:

- Namespace axis and namespace node type. The namespace axis returns the actual XML namespace declaration nodes for a particular element. You can therefore manipulate XML prefix or URI declarations within an XPath expression. This axis returns an empty node set for bit streams that are not XML.
- If you use the **`id()`** function, it throws an MbRecoverableException.

The `evaluateXPath()` method can be called on a MbMessage object (for absolute paths), or on a MbElement object (for relative paths). The XPath expression is passed to the method as a string parameter. A second form of this method is provided that takes an MbXPath object. This object encapsulates an XPath expression along with variable bindings and namespace mappings, if these are required.

The `evaluateXPath()` method returns an object of one of these four types, depending on the expression return type:

- java.lang.Boolean, representing the XPath Boolean type
- java.lang.Double, representing the XPath number type
- java.lang.String, representing the XPath string type
- java.util.List, representing the XPath node set. The List interface represents an ordered sequence of objects, in this case MbElements. It allows direct access to the elements, or the ability to get an Iterator or an MbElement array.

XPath variable binding

XPath 1.0 supports the ability to refer to variables that have been assigned before the expression that contains them is evaluated. The MbXPath class has three methods for assigning and removing these variable bindings from user Java code. The value must be one of the four XPath 1.0 supported types:

- Boolean
- node set
- number
- string

XPath namespace support

For XML messages, namespaces are referred to by using a mapping from an abbreviated namespace prefix to the full namespace URI, as shown in the following XML example:

```
<ns1:aaa xmlns:ns1='http://mydomain.com/namespace1'
         xmlns:ns2='http://mydomain.com/namespace2'>
  <ns2:aaa>
    <ns1:bbb/>
  </ns2:aaa>
</ns1:aaa>
```

The namespace prefix is convenient for representing the namespace, but is meaningful only within the document that defines that mapping. The namespace URI defines the global meaning. Also, the concept of a namespace prefix is not meaningful for documents that are generated in a message flow, because a namespace URI can be assigned to a syntax element without an XMLNS mapping having been defined.

For this reason, the XMLNSC and MRM parsers expose only the namespace URI to the integration node and to user code (ESQL or user-defined code). By using ESQL, you can set up your own mappings to

create abbreviations to these potentially long URIs. These mappings are not related in any way to the prefixes that are defined in the XML document (although they can be the same name).

By using the XPath processor you can map namespace abbreviations on to URIs that are expanded at evaluation time. The MbXPath class contains methods to assign and remove these namespace mappings. For example:

```
MbNamespaceBindings ns = new MbNamespaceBindings();
ns.addBinding("other", "http://mydomain.com/namespace2");
ns.setDefaultNamespace("http://mydomain.com/namespace2");
MbXPath xp = new MbXPath("/aaa/other:aaa/bbb", ns);
List<MbElement> nodeset = (List<MbElement>)message.evaluateXPath(xp);
```

Updating a message by using XPath extensions

The XPath implementation in IBM App Connect Enterprise provides the following extra functions for modifying the message tree:

set-local-name(*object*)
Sets the local part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-namespace-uri(*object*)
Sets the namespace URI part of the expanded name of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

set-value(*object*)
This function sets the string value of the context node to the value specified in the argument. *object* can be any valid expression, and is converted to a string as if a call to the string function is used.

To allow for syntax element trees to be built as well as modified, the following axis is available in addition to the 13 that are defined in the XPath 1.0 specification:

select-or-create::*name* or ?*name*
?*name* is equivalent to select-or-create::*name*. If *name* is @name, an attribute is created or selected. This selects child nodes matching the specified name, or creates new nodes according to the following rules:

- ?*name* selects children called *name* if they exist. If a child called *name* does not exist, ?*name* creates it as the last child, then selects it.
- ?$*name* creates *name* as the last child, then selects it.
- ?^*name* creates *name* as the first child, then selects it.
- ?>*name* creates *name* as the next sibling, then selects it.
- ?<*name* creates *name* as the previous sibling, then selects it.

Using JAXB with a JavaCompute node
Java Architecture for XML Binding (JAXB) enables Java applications to work with a model of message data as Java object classes. This Java object model can be accessed and manipulated by using getter and setter methods.

About this task

Developing messaging solutions by using JAXB programming provides an alternative to using the IBM App Connect Enterprise Java plug-in API, or the Document Object Model API, and does not require detailed knowledge of the IBM App Connect Enterprise tree representation of the message data.

You can use Java Architecture for XML Binding (JAXB) with a JavaCompute node to process your messages by accessing, creating, and manipulating JAXB Java object classes that you generate from your message model schema files. JAXB Java object classes are a Java object representation of your message, and can be used with Java code completion.

The following topics describe how to use Java Architecture for XML Binding (JAXB) in your IBM App Connect Enterprise messaging solutions:

- “Creating a Java project” on page 1773
- “Creating JAXB Java object classes” on page 1773
- “Processing message body data by using JAXB Java object classes” on page 1775

Creating a Java project
Java projects are used in the IBM App Connect Enterprise Toolkit to contain Java object classes.

About this task

To create a Java project in the Integration Development perspective, complete the following steps:

Procedure

1. Click **File** > **New** > **Other**.
   A window opens in which you can select a wizard.
2. Expand **Java**, select **Java Project**, and click **Next**.
   The **New Java Project** wizard opens.
3. Enter a name for your data design project, and then click **Finish**.
   The **Open associated perspective** dialog is displayed.
4. Click **No**.
   Your Java project is created, and is displayed in the Application Development view, under **Independent resources**.

What to do next

Before you can use your Java project in a messaging solution, you must complete one of the following tasks:

- Include your Java project in an application or library; see “Adding a project to an application, integration service, or library” on page 1979.
- Refer to your Java project from an integration project; see “Referencing resources in other libraries” on page 1976.

After you include your Java Project in an application, library, or integration project, you can complete the following task:

- Create Java object classes by using the JAXB 2.0 schema compiler that is provided with IBM App Connect Enterprise; see “Creating JAXB Java object classes by using the JAXB 2.0 schema compiler” on page 1774.

Creating JAXB Java object classes
Java Architecture for XML Binding (JAXB) Java object classes provide a Java object representation of your message data, and can be used in JavaCompute nodes to build messaging solutions. JAXB support in IBM App Connect Enterprise includes Java code completion for discovery of the object model, getter methods, and setter methods.

About this task

You can create JAXB Java object classes from your message model schema files by using the JAXB 2.0 schema compiler that is provided with IBM App Connect Enterprise.

If you want to use JAXB Java object classes that you generated by using a JAXB 2.0 schema compiler outside of IBM App Connect Enterprise, you must reference them from your application or integration project before you can use them in a JavaCompute node; see “Adding Java code dependencies” on page 1759.

The following topics describe how to create JAXB Java object classes by using IBM App Connect Enterprise Toolkit:

- Creating JAXB Java object classes by using a wizard
- Creating JAXB Java object classes by using the command line

Creating JAXB Java object classes by using a wizard
Create Java Architecture for XML Binding (JAXB) Java object classes by using the **Generate JAXB Java object classes** wizard.

Before you begin

- You must create a message model schema file; see "Constructing message models" on page 2133.

About this task

You can create your JAXB Java object classes when you create a Java class for a JavaCompute node in your message flow. Alternatively, you can create your JAXB Java object classes first, to be used in one or more JavaCompute node Java classes later. The same **Generate JAXB Java object classes** wizard is used whether you call the wizard interdependently or from within the **New Java Compute Node Class** wizard.

To generate JAXB Java object classes from your message model schema files by using the **Generate JAXB Java object classes** wizard, complete the following steps:

Procedure

1. Optional: In the Application Development view, click the view menu (shown by a white arrow in the Application Development view toolbar), and select **Hide Namespaces**.

   This step is required if you want to create your JAXB Java object classes from a folder that contains multiple message model schema files.
2. Right-click the message model schema file that you want to use, and click **Generate JAXB Java object classes**. To create JAXB Java object classes from multiple message model schema files at the same time, right-click a folder that contains the message model schema files that you want to use, and click **Generate JAXB Java object classes**.

   The **Generate JAXB Java object classes** wizard is displayed.
3. Complete the fields as required, then click **Finish**.

Results

Your JAXB Java object classes are visible in the Application Development view, under the **src** folder of the Java project that you selected in the wizard.

What to do next

- For information about how to process message body data by using your JAXB Java object classes, see "Processing message body data by using JAXB Java object classes" on page 1775.

Creating JAXB Java object classes by using the JAXB 2.0 schema compiler
Create Java Architecture for XML Binding (JAXB) Java object classes by using the JAXB 2.0 schema compiler from a command window.

Before you begin

- You must create a message model schema file; see "Constructing message models" on page 2133.
- You might need to add the `bin` folder of the IBM Java SDK to your system path. The `bin` folder of the IBM Java SDK that is included with IBM App Connect Enterprise at *`install_dir`*`\common\jdk\bin`.

About this task

To generate JAXB Java object classes from your message model schema files by using an IBM App Connect Enterprise command window, complete the following steps:

Procedure

1. In the Application Development view, right-click the message model schema file that you want to use to generate your JAXB Java object classes and select **Properties**.

   A window opens that shows the properties for your selected message model schema file.
2. Note the *Location* of your message model schema file, then click **OK** to close the properties window.
3. In the Application Development view, expand the Java project that you want to contain your JAXB Java object classes, then Right-click the **src** folder and select **Properties**.

   A window opens that shows the properties for your selected **src** folder.
4. Note the *Location* of your selected **src** folder, then click **OK** to close the properties window.
5. Open an IBM App Connect Enterprise command window. If you have not added the `bin` folder of the IBM Java SDK to your system path, change directory to this folder. The `bin` folder of the IBM Java SDK that is included with the IBM App Connect Enterprise Toolkit is located at *`install_dir`*`\common\jdk\bin`
6. Optional: To view a full list of commands for the JAXB 2.0 schema compiler, enter the following command:

   ```
   xjc -help
   ```
7. Enter the following command to create your JAXB Java object classes:

   ```
   xjc MessageModelSchema -d JavaProject -p JavaPackage
   ```

   Where *MessageModelSchema* is the location of your message model schema file, *JavaProject* is the location of the **src** folder of your Java project, and *JavaPackage* is the Java package where the generated classes are placed.

   The JAXB 2.0 schema compiler generates JAXB Java object classes from your message model schema file, and saves them to the **src** folder of the Java project that you specified in the command.
8. In the IBM App Connect Enterprise Toolkit, open the **Package Explorer** view:

   a) Click **Window** > **Show view** > **Other...**

   b) In the **Show View** window, expand **Java** and select **Package Explorer**, then click **OK**.
9. In the **Package Explorer** view, right-click the Java project that you selected to contain your JAXB Java object classes and select **Refresh**.

Results

Your JAXB Java object classes are visible in the Application Development view, under the **src** folder of your Java project.

What to do next

- For information about how to process message body data by using your JAXB Java object classes, see “Processing message body data by using JAXB Java object classes” on page 1775.

Processing message body data by using JAXB Java object classes

Use the JAXB object classes generated by a Java Architecture for XML Binding (JAXB) compiler by using the JAXB class template.

Before you begin

- Create your JAXB Java object classes; see “Creating JAXB Java object classes” on page 1773.

- Add a JavaCompute node to your message flow; see node.

About this task

You can use Java Architecture for XML Binding (JAXB) with a JavaCompute node to process your messages by accessing, creating, and manipulating JAXB Java object classes that you generate from your message model schema files.

JAXB Java object classes are a Java object representation of your message, and can be used with Java code completion. The **Process via JAXB class** template in the **New Java Compute Node Class** wizard generates template code for processing your messages by using JAXB Java object classes. If you also used the **New Java Compute Node Class** wizard either to generate your JAXB Java object classes, or to reference your existing JAXB Java object classes, step 1 might be completed for you.

To process message body data by using JAXB Java object classes, complete the following steps:

Procedure

1. Optional: Modify the `onInitialize ()` method to reference the package or packages that contain your Java object binding classes.

   ```
   public void onInitialize() throws MbException {
    try {
     // TODO Update context path "com.example.jaxb" to be the package of your
     // Java object classes that were generated by a Java Architecture for XML
     // Binding (JAXB) binding compiler
     jxbcntxt = JAXBContext.newInstance("com.example.jaxb");
   ```

   Replace `com.example.jaxb` with your JAXB Java object bindings classes package.

   Note: When you deploy or restart your message flow, the `onInitialize ()` method is called only once, and initializes JAXB context for all processing threads.

   If an error is thrown by the generation of the JAXB context, the code that is provided in the "Process via JAXB class" template throws an `MbUserException`, and the deployment operation fails. If the deployment operation fails, inspect the error message that is recorded in the host system event log to determine the cause. The most likely cause is that one or more package names cannot be resolved, or that one or more of the packages you specified do not contain the required JAXB object factory or bindings properties.

2. Modify the `evaluate()` method to process each message that is passed in a `MbMessageAssembly` object. The `MbMessageAssembly` object is defined in the Java user-defined node API.

   a) The first section of the `evaluate()` method extracts the message data from the assembly, creates an output assembly and copies message headers.

   ```
   public void onInitialize() throws MbException {
   try {
   // TODO Update context path "com.example.jaxb" to be the package of your
   // Java object classes that were generated by a Java Architecture for XML
   // Binding (JAXB) binding compiler
   jaxbContext = JAXBContext.newInstance("com.example.jaxb");
   } catch (JAXBException e) {
   // This exception will cause the deploy of this Java compute node to fail
   // Typical cause is the JAXB package above is not available
   MbUserException mbue = new MbUserException(this, "onInitialize()",
   "", "", e.toString(), null);
   throw mbue;
   }
   }
   ```

   b) The next part in the `evaluate()` method *unmarshals* the input message data into the set of Java objects that are bound from the message schema. Unmarshalling creates a Java object model copy of the message data that is in memory.

   ```
   try {
   // unmarshal the input message data from the message tree into your Java object classes
   ```

```
Object inMsgJavaObj =
 jaxbContext.createUnmarshaller().unmarshal(inMessage.getDOMDocument());
```

c) The next part in the `evaluate()` method is where you insert your own JAXB code to create or update the output message by processing message data. See external resources for details of how to program transformations by using JAXB Java object classes. For information about debugging in Java, see Java Debugger.

```
// ------------------------------------------------------
// Add user code below to build the new output data by updating
// your Java objects or building new Java objects
Object outMsgJavaObj = inMsgJavaObj;
// End of user Java object processing
// ------------------------------------------------------
```

The following example shows some JAXB code that adds a record to a CSV message, and sets the value of the three fields in that record:

```
// Example: add a record of fixed data
CsvMsg outMsgJavaObj = (CsvMsg) inMsgJavaObj;
Record additionalCsvRecord = new Record();
additionalCsvRecord.setField1("My new field 1 text");
additionalCsvRecord.setField2("My new field 2 text");
additionalCsvRecord.setField3("My new field 3 text");
outMsgJavaObj.getRecord().add(additionalCsvRecord);
// end example
```

d) The next part in the `evaluate()` method *marshals* the processed Java object classes back into the message tree. Modify this code to pass the relevant Java object in place of the template default. If your JAXB transformation code updated the input message, pass the Java object class that was created during unmarshalling into the marshalling code. If you are creating an output message, pass the Java object class for your new output message into the marshalling code.

The template sets the output domain to XMLNSC; if you require the output message to be in the DFDL or SOAP domains, you must modify this code.

```
// TODO set the required
 domain for the output message, eg XMLNSC
Document outDocument = outMessage.createDOMDocument(MbXMLNSC.PARSER_NAME);
// marshal the new or updated output Java object class into the message tree
jaxbContext.createMarshaller().marshal(outMsgJavaObj, outDocument);
```

What to do next

- You can use the Flow Exerciser to check that your message flow processes messages as expected; see “Testing your message flow by using the Flow Exerciser” on page 648.

Manipulating other parts of the message tree by using a JavaCompute node

You can access parts of the message tree other than the message body data. These parts of the logical tree are independent of the domain in which the message exists, and all these topics apply to messages in all supported domains, including the BLOB domain. You can access all parts of the message tree by using a JavaCompute node.

About this task

Elements of the message tree can be accessed in the same way as the message body data, by using a JavaCompute node.

- “Accessing headers by using a JavaCompute node” on page 1778
- “Updating the local environment with the JavaCompute node” on page 1779
- “Updating the Global Environment with the JavaCompute node” on page 1779

Accessing headers by using a JavaCompute node
You can access headers by using a JavaCompute node.

About this task

If an input node receives an input message that includes message headers that the input node recognizes, the node starts the correct parser for each header. Parsers are supplied for most IBM MQ headers. The following topics provide guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

- "Copying message headers by using a JavaCompute node" on page 1778
- "Accessing the MQMD header by using a JavaCompute node" on page 1778
- "Accessing the MQRFH2 header by using a JavaCompute node" on page 1778

For further details of the contents of these and other IBM MQ headers for which IBM App Connect Enterprise provides a parser, see Element definitions for message parsers.

Copying message headers by using a JavaCompute node
You can copy message headers by using a JavaCompute node.

The *Modifying Message Class* template in the JavaCompute node wizard generates the following code to copy message headers using a JavaCompute node:

```
public void copyMessageHeaders(MbMessage inMessage, MbMessage outMessage) throws MbException
{
 MbElement outRoot = outMessage.getRootElement();
 MbElement header = inMessage.getRootElement().getFirstChild();

 while(header != null && header.getNextSibling() != null)
 {
  outRoot.addAsLastChild(header.copy());
  header = header.getNextSibling();
 }
}
```

Accessing the MQMD header by using a JavaCompute node
IBM MQ and WebSphere MQ Everyplace messages include an MQMD header. You can use a JavaCompute node to refer to the fields in the MQMD, and to update them.

About this task

The following Java code shows how to add an MQMD header to your message:

```
public void addMqmd(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();

 // create a top level 'parser' element with parser class name
 MbElement mqmd = root.createElementAsFirstChild("MQHMD");

 // specify next parser in chain
 mqmd.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
  "Format",
  "XMLNS");
}
```

Accessing the MQRFH2 header by using a JavaCompute node
You can use a JavaCompute node to add an MQRFH2 header to an outgoing message.

About this task

When you construct MQRFH2 headers in a JavaCompute node, two types of field exist:

- Fields in the MQRFH2 header structure (for example, Format and NameValueCCSID)
- Fields in the MQRFH2 NameValue buffer (for example mcd and psc)

The following code adds an MQRFH2 header to an outgoing message that is to be used to make a subscription request:

```
public void addRfh2(MbMessage msg) throws MbException
{
 MbElement root = msg.getRootElement();
 MbElement body = root.getLastChild();

 // insert new header before the message body
 MbElement rfh2 = body.createElementBefore("MQHRF2");

 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Version", new Integer(2));
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Format", "MQSTR");
 rfh2.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "NameValueCCSID", new Integer(1208));

 MbElement psc = rfh2.createElementAsFirstChild(MbElement.TYPE_NAME, "psc", null);
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "Topic", "department");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QMgrName", "QM1");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "QName", "PUBOUT");
 psc.createElementAsFirstChild(MbElement.TYPE_NAME, "RegOpt", "PersAsPub");

 MbXPath xp = new MbXPath("/MQMD/Format" + "[set-value(´MQHRF2´)]", root);
 root.evaluateXPath(xp);
}
```

In this example, the MQHRF2 parameter is the parser class name, which is different from the parser element name (MQRFH2). For a list of the parsers, root element names, and class names for different headers, see Available parsers.

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

About this task

The following information shows how to update the local environment:

Procedure

1. Make a new copy of the local environment to update it. Use the full version of the copy constructor to create a new MbMessageAssembly object, as shown in the following example:

```
MbMessage env = assembly.getLocalEnvironment();
MbMessage newEnv = new MbMessage(env);

newEnv.getRootElement().createElementAsFirstChild(
  MbElement.TYPE_NAME_VALUE,
  "Status",
  "Success");

MbMessageAssembly outAssembly = new MbMessageAssembly(
  assembly,
  newEnv,
  assembly.getExceptionList(),
  assembly.getMessage());

getOutputTerminal("out").propagate(outAssembly);
```

2. Edit the copy to update the local environment.

Updating the Global Environment with the JavaCompute node
The Global Environment tree is always created when the logical tree is created for an input message. However, the message flow neither populates it nor uses its contents. You can use this tree for your own

purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

About this task

The Global Environment can be altered across the message flow, therefore do not make a copy of it to alter. The following Java code shows how to alter the Global Environment:

```
MbMessage env = assembly.getGlobalEnvironment();
env.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME_VALUE, "Status", "Success");

getOutputTerminal("out").propagate(assembly);
```

Accessing the ExceptionList tree using Java
The ExceptionList tree is created with the logical tree when an input message is parsed.

About this task

The tree is initially empty, and is only populated if an exception occurs during message flow processing. It is possible that more than one exception can occur; if more than one exception occurs, the ExceptionList tree contains a subtree for each exception.

You can access the ExceptionList tree in JavaCompute nodes from the `MbMessageAssembly` parameter of your `evaluate()` method.

You can access the ExceptionList tree in a node in an error handling procedure. For example, you might want to route the message to a different path based on the type of exception.

You can use the querying capabilities within XPath to carry out this task. The descendant axis (`//`) of XPath gives you the ability to search for an element by name regardless of its depth in the tree. For example:

```
    //ParserException
```

returns all elements in the tree named `ParserException`.

If you are specifically interested in a particular message, you can use a predicate (see predicates for further information) to narrow down the result set. For example:

```
    //ParserException[Number=5016]
```

returns only the exception that contains `Number=5016`.

If you only want to extract the text message associated with this exception, the following XPath expression returns this as a `java.lang.String`:

```
    string(//ParserException[Number=5016]/Text)
```

The following Java code extracts this text from your code:

```
String text =
(String)inAssembly.getExceptionList().evaluateXPath("string(//ParserException[Number=5016]/Text)");
```

For information on accessing the ExceptionList tree using ESQL, see “Accessing the ExceptionList tree using ESQL” on page 1658.

Accessing the global cache by using a JavaCompute node
You can use a JavaCompute node to interact with a map in a global cache or an external WebSphere eXtreme Scale grid.

Before you begin

You can also access the global cache by using a Mapping node; see Accessing the global cache by using a Mapping node.

- Read the concept information in "Data caching overview" on page 272.
- Ensure that the global cache or external grid is available. The cache is disabled by default. To use the cache, configure it as described in "Configuring the embedded global cache" on page 285.
- If you are connecting to an external grid, create a WXSServer policy to define how the connection is made. See "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.
- To use your own Java classes with the global cache, put the JAR files that contain the Java classes in one of the shared-classes directories. These Java classes must be available to all integration servers that participate in the global cache. For more information about the shared-classes directories, see "Java shared classloader" on page 1752.

About this task

You can use a JavaCompute node to store data in a global map. You can then create another JavaCompute node that can get data from the global cache for processing or routing. The JavaCompute node uses the MbGlobalMap object to access the global cache or external grid. This class can be used to get a map, and to put or get data in a map. You cannot create a map explicitly, but if you get a map that does not exist, the map is created automatically. When you get a global map from an external grid, the getGlobalMap method makes a connection to the grid if one does not exist.

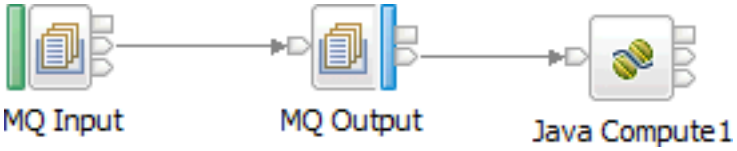
Interactions with the cache happen outside the message flow transaction, and are committed immediately. If an exception is thrown downstream of the node that interacts with the cache, the cache interactions are not rolled back.

The following steps describe how one message flow adds data to a map, and another flow gets the data from that map.

Procedure

1. Create a message flow that contains a JavaCompute node.

   You can place the JavaCompute node before or after an MQOutput node. In the following example, the JavaCompute node is placed after an MQOutput node so that the JavaCompute node can use the message ID of the output message.

2. Double-click the JavaCompute node.

   The **New Java Compute Node Class** wizard opens.

3. On the **Java Compute Node Class** pane, provide the appropriate information, then click **Next**.

   You can accept the default values by clicking **Next**.

4. On the **Java Compute Node Class Template** pane, select the appropriate template for your scenario, then click **Finish**.

   The template Java file opens in the editor, indicating where you can add your own code.

5. To get a map from the embedded cache, or create the map if it does not exist, add the MbGlobalMap object, as shown in the following example.

   The MbGlobalMap object is also described in the Java Plugin API documentation.

   ```
   MbGlobalMap globalMap = MbGlobalMap.getGlobalMap("MyMap");
   ```

   This example gets the map called "MyMap". If the map does not exist, it is created. Alternatively, when you are using the embedded cache, you can use the default map by not specifying a map name, as shown in the following example. When you are connecting to an external grid, no default map exists. You must specify a valid map name on the external grid.

   ```
   MbGlobalMap globalMap = MbGlobalMap.getGlobalMap();
   ```

   The cache uses WebSphere eXtreme Scale dynamic maps. Any map name is allowed, apart from names that begin with SYSTEM.BROKER, which is reserved for use by the integration node. The default map is named SYSTEM.BROKER.DEFAULTMAP; you can use or clear this map.

   To get a map from an external grid, add the MbGlobalMap object, specifying the name of the map on the external grid and the name of the policy that is used to connect to the grid:

   ```
   MbGlobalMap remoteCacheServerMap =
    MbGlobalMap.getGlobalMap("MyMap.LUT","remoteCacheServer");
   ```

   In this example, the map, `MyMap.LUT`, is on an external grid. The policy that is used to connect to the grid is called "remoteCacheServer".
6. Optional: Specify how long, in seconds, the data remains in the global cache before it is removed automatically.

   This time is known as the *time to live* and is counted from when that map entry is last updated. You specify this time when you get an MbGlobalMap object. The value applies to all cache entries that are created by using that MbGlobalMap object. For detailed instructions, see “Specifying how long data remains in the global cache by using the JavaCompute node” on page 1783.
7. To put data in the map, you must create a key and value pair in the map, as shown in the following example.

   For keys and values, Java primitive types and strings are supported. Java objects are supported as values. For the embedded grid only, keys can also be arrays of primitives or strings. (To use your own Java classes with the global cache, put the JAR files that contain the Java classes in one of the shared-classes directories.)

   ```
   globalMap.put(key, val);
   ```
8. Save the message flow.
9. To access the stored data from another message flow, create a flow that contains a JavaCompute node.

   For example, the following message flow contains a JavaCompute node that retrieves the appropriate information from the map, then sends a response message through an MQReply node.

10. Repeat steps 2, 3, and 4 to create a Java file for your new JavaCompute node.
11. Use the MbGlobalMap object to get the map that you created in step 5, then get the data that you added to the map in step 6. If you know that the data that you are retrieving is a string, for example, specify string when you get the data, as shown in the following example.

    ```
    MbGlobalMap globalMap = MbGlobalMap.getGlobalMap("MyMap");
    ...
    ```

```
String val = (String)globalMap.get(key);
```

The following example shows how to get the data that you added to a map on an external grid:

```
MbGlobalMap remoteCacheServerMap =
 MbGlobalMap.getGlobalMap("MyMap.LUT","remoteCacheServer");
...
String val = (String)remoteCacheServerMap.get(key);
```

To prevent the cache from becoming too big, when you have finished with a key and value pair, use the `remove()` method to delete that data from the map.

You can use content assist to list available options when you are constructing Java code. To access content assist, place the cursor at the point of insertion, then either click **Edit** > **Content Assist** or press Ctrl+Space. For example, if you type "myMap", then press Ctrl+Space, you can choose from a list of options, including the following actions.

- *remove*, which removes a single object from a global cache
- *containsKey*, which indicates whether a key exists in a map
- *update*, which updates an existing object

All available options are documented in the Javadoc for the Java Plugin API (see IBM Integration API).

12. Save your message flow.
13. Deploy your message flows.

Specifying how long data remains in the global cache by using the JavaCompute node
Specify when data is removed automatically from the global cache.

Before you begin

Create a message flow that contains a Mapping node or a JavaCompute node that is configured to access the global cache. For more information, see Accessing the global cache by using a Mapping node or “Accessing the global cache by using a JavaCompute node” on page 1781.

About this task

When you get an MbGlobalMap object, you can specify how long the data remains in the global cache before it is removed automatically. This time is known as the *time to live* and is counted from when that map entry is last updated. The value applies to all cache entries that are created by using that MbGlobalMap object in that instance of the message flow. Data that is already in the map that you specify, or that is created by another MbGlobalMap object, is not affected by the time to live value.

By default, the time to live is set to zero so that the data is never removed. To set a specific time to live, create a session policy, which you can reference from the MbGlobalMap object.

You can specify the time to live value in two ways. The following examples both demonstrate how to set a time to live of 60 seconds for cache entries in the map "myMap".

Example 1

```
MbGlobalMap myMap = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(60));
```

Example 2

```
MbGlobalMapSessionPolicy sessionPol = new MbGlobalMapSessionPolicy(60);
MbGlobalMap myMap = MbGlobalMap.getGlobalMap("myMap", sessionPol);
```

Data in the map "myMap" is removed automatically 60 seconds after it is last updated.

You can create multiple MbGlobalMap objects in different flows, integration servers, or integration nodes, all resolving to the same map in the global cache, but with different time to live values. However, you must

structure the code in a specific way when you want to put multiple cache entries, with different time to live values, into the same map in the same message flow.

In the following example, the MbGlobalMap objects are all created before the put statements are made. All three MbGlobalMap objects resolve to the same underlying map. The time to live value of each subsequent MbGlobalMap object replaces the value that was set for the previous object. Therefore, for each put statement, data is removed after 20 seconds.

```
MbGlobalMap m1 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(60));
MbGlobalMap m2 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(40));
MbGlobalMap m3 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(20));
m1.put("k1","v1");
m2.put("k2","v2");
m3.put("k3","v3");
```

In the following example, each put statement is made directly after the associated MbGlobalMap object is created. Therefore, the cache entry that is made by each put statement assumes a different time to live value.

```
MbGlobalMap m1 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(60));
m1.put("k1","v1");
m1 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(40));
m1.put("k2","v2");
m1 = MbGlobalMap.getGlobalMap("myMap", new MbGlobalMapSessionPolicy(20));
m1.put("k3","v3");
```

Accessing message flow user-defined properties from a JavaCompute node
Customize a JavaCompute node to access properties that you have associated with the message flow in which the node is included.

About this task

To access these properties from a JavaCompute node, use the `getUserDefinedAttribute(`*name*`)` method, where *name* is the name of the property that you are accessing. The type of the object that is returned depends on the type of the property that you are accessing. The object has one of a set of types:

- MbDate
- MbTime
- MbTimestamp
- Boolean
- byte[]
- String
- Integer 32-bit values
- Long 64-bit values
- Double
- BigDecimal
- BitSet

You cannot access user-defined properties in the constructor. To access them at initialization time, implement the following method, and use it to access the user-defined properties.

```
public void onInitialize() throws MbException
{
   // access the user-defined properties here
}
```

You can use the administration REST API to modify the values of user-defined properties at run time, as described in “Setting message flow user-defined properties at run time by using the administration REST API” on page 436.

You can use the IBM Integration API to change the value of user-defined properties. The `getUserDefinedAttribute()` method is not supported for User-Defined Nodes (UDN). Use the `getUserDefinedPropertyNames()`, `getUserDefinedProperty()`, and `setUserDefinedProperty()` methods to query, discover, and set user-defined properties.

Accessing a user-defined policy from a JavaCompute node
Use a JavaCompute node to query properties dynamically at run time, in user-defined policies.

Before you begin

Complete the following tasks:

- "Creating a message flow" on page 574
- User Defined policy (UserDefined)

About this task

If you created a user-defined policy, and created properties for that policy, you can query those properties in a JavaCompute node. For example, you can create a user-defined policy to set timeouts for processing HTTP messages.

Procedure

1. Right-click the JavaCompute node and click **Open Java** to create and open a Java file in the Editor view, or open an existing file.
2. Create the Java class for the JavaCompute node from which you want to access the user-defined policy.
3. Add user code as shown below to the Java class to define the user-defined policy. In the example shown, a user-defined policy called MyUDCS has been defined in a policy project called PP1.

```
MbPolicy myPol = getPolicy("UserDefined", "{PP1}:MyUDCS");
```

4. Use the following code to access the policy.

```
if (myPol != null) {
System.out.println("Found policy: "+myPol.getName());
System.out.println(" with type: "+myPol.getType());
System.out.println(" and properties:");
System.out.println(" prop1: "+myPol.getPropertyValueAsString("prop1"));
System.out.println(" prop2: "+myPol.getPropertyValueAsString("prop2"));
System.out.println(" prop3: "+myPol.getPropertyValueAsString("prop3"));
}
```

   This lookup syntax can be qualified by policy project. If no policy project is qualified, the policy project that is defined on the *defaultPolicyProject* property in the `node.conf.yaml` configuration file is used. Null is returned if either the property or the policy does not exist.

   If you created a user-defined policy, and created properties for that policy, you can query those properties by using a Java class method that can be called from ESQL.

Interacting with databases by using the JavaCompute node
Access databases from Java code included in the JavaCompute node.

About this task

Choose from the following options for database interaction:

- MbSQLStatement
- JDBC API in an unmanaged environment
- SQLJ

If you use JDBCProvider for type 4 connections or MbSQLStatement, the databases that you access can participate in globally coordinated transactions. In all other cases, database access cannot be globally coordinated.

MbSQLStatement

About this task

The MbSQLStatement class provides full transactional database access by using ESQL and ODBC. Global coordination is provided by IBM MQ on distributed systems, and by RRS on z/OS.

Create instances of the MbSQLStatement class by using the createSQLStatement() method of MbNode, passing to the method the ODBC data source, an EQSL statement, and, optionally, the transaction mode.

- Calling select() on this object returns the results of the query.
- Calling execute() on this object runs a query where no results are returned, such as updating a table.

The following Java code shows how to access a database by using MbSQLStatement:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

String table = "dbTable";

MbSQLStatement state = createSQLStatement( "dbName",
 "SET OutputRoot.XMLNS.integer[] = PASSTHRU('SELECT * FROM " + table + "');" );

state.setThrowExceptionOnDatabaseError(false);
state.setTreatWarningsAsErrors(true);
state.select( assembly, newAssembly );

int sqlCode = state.getSQLCode();
if(sqlCode != 0)
{
 // Do error handling here
}

getOutputTerminal("out").propagate(assembly);
```

JDBC API in an unmanaged environment

About this task

You can access standard Java APIs in the code that you write for your JavaCompute nodes, including JDBC calls. You can therefore use JDBC APIs to connect to a database, write to or read from the database, and disconnect from the database. On operating systems other than z/OS, your JDBC connection code can call both type 2 and type 4 JDBC drivers in this environment, but they are not supplied. You must obtain these drivers from your database vendor. On z/OS, type 2 drivers are not supported.

If you choose this method to access databases, managing the transactions is not supported; your code must manage the local commit and rollback of database changes. Your code must also manage the connection lifecycle, connection thread affinity, and connection pooling. You must also monitor the access to databases when you use this technique to ensure that these connections do not cause interference with connections made by the integration server. In particular, be aware that type 2 drivers bridge to an ODBC connection that might be in use in message flows that access databases from ESQL.

SQLJ

About this task

SQLJ is a Java extension that you can use to embed static SQL statements within Java code. Create SQLJ files by using the IBM App Connect Enterprise Toolkit. The resource manager does not coordinate database access when using SQLJ.

Procedure

1. Enable SQLJ capability in the IBM App Connect Enterprise Toolkit:
   a) Select **Window** > **Preferences**.
   b) Expand **General**.
   c) Select **Capabilities**.
   d) Select **Data**.
   e) Click **OK**.
2. Create an SQLJ file within a Java project:
   a) Right-click the ***Java project*** in which you want to create the file.
   b) Select **New** > **Other**.
   c) Expand **Data**.
   d) Expand **SQLJ Applications**.
   e) Select **SQLJ File**.
   f) Click **Next**.
   g) Follow the directions given by the **New SQLJ File** wizard to generate the SQLJ file.

Results

You can now reference the class in this SQLJ file from a JavaCompute node class in this project or in another referenced project.

Calling an Enterprise Java Bean
You can call an Enterprise Java Bean (EJB) from a JavaCompute node.

Before you begin

- Ensure that all required Java classes are in the IBM App Connect Enterprise shared-classes directories, or are referenced in the CLASSPATH environment variable. You can use the wildcard character (*) at the end of a directory path specifier to load all JARs in that directory path.
- Ensure that the user JAR files that are needed for EJB access are referenced in CLASSPATH. For more information, see the documentation for the application server that is hosting the EJB.

Example

The following example shows how to call an EJB from a JavaCompute node:

```
public class CallAckNoAckEJB_JavaCompute extends MbJavaComputeNode {

   public void evaluate(MbMessageAssembly inAssembly) throws MbException {
                MbOutputTerminal out = getOutputTerminal("out");
                MbOutputTerminal alt = getOutputTerminal("alternate");

                MbMessage inMessage = inAssembly.getMessage();

                // create new message
                MbMessage outMessage = new MbMessage(inMessage);
                MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,outMessage);

                try {
                        // ----------------------------------------------------------
                        // Add user code below

                        String response = null;
                        String responseMessage = null;

                        Properties properties = new Properties();
                        properties.put(Context.PROVIDER_URL, "iiop://localhost:2809");
                        properties.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.websphere.naming.
WsnInitialContextFactory");

                        try {
```

```
                           Context initialContext = new InitialContext(properties);
                                Object obj = initialContext.lookup("ejb/com/acme/ejbs/AckNoAckHome");
                                AckNoAckHome ejbHome = (AckNoAckHome)javax.rmi.PortableRemoteObject.
narrow(obj,AckNoAckHome.class);

                                AckNoAck ackNoAck = ejbHome.create();
                                responseMessage = ackNoAck.getAck();
                                response = "Ack";
                           } catch(Exception e) {
                                responseMessage = e.getMessage();
                                response = "NoAck";
                           }

                           MbElement cursor = outMessage.getRootElement().getFirstElementByPath("/XML/
AckNoAck");
                           cursor.createElementAsLastChild(MbElement.TYPE_NAME,"Response",null);

 cursor.getLastChild().createElementAsLastChild(MbElement.TYPE_NAME,response,null);

 cursor.getLastChild().getLastChild().createElementAsLastChild(MbElement.TYPE_VALUE,null,
responseMessage);

                           // End of user code
                           // ------------------------------------------------------------

                           // The following should only be changed
                           // if not propagating message to the 'out' terminal
                           out.propagate(outAssembly);

                } finally {
                           // clear the outMessage
                           outMessage.clearMessage();
                }
   }
}
```

JavaCompute node exception handling
The `evaluate()` method throws an `MbException`.

If your code throws other classes of checked exceptions, they must be caught and rethrown as `MbException`. Typically, you use the `MbUserException` class, as shown in the example code. `MbUserException` is a subclass of `MbException`, and has a public constructor. IBM App Connect Enterprise utility functions can throw other subclasses of MbException that have private constructors.

About this task

Exceptions can occur during message flow processing (during the `evaluate()` method) and during the `onInitialize()` method. If you have implemented the `onInitialize()` method, and you detect an unrecoverable error in the node configuration, an exception of class `MBException` is issued and the flow fails to initialize. Either your flow fails to deploy, and error message BIP4157 is issued, or the flow does not start and does not appear in the list of running flows.

Do not run code in an `onInitialize()` method that depends on another external resource that might not be there because the calls to that method will not be retried. If you need to initialize an external connection, for example, initialize it during the first call to the `evaluate()` method. If the initialization does not work, you can throw an exception that will be handled like any other flow exception.

The following example code shows a simple implementation of a JavaCompute node that routes a CSV message to the `out` or `alt` terminals, based on a header field that contains a positive or negative number. If the text in the field is not numeric, a `NumberFormatException` is thrown. The example code shows how this exception can be caught and thrown as a `MbUserException` so that the handling of `MbException` built into IBM App Connect Enterprise from JavaCompute nodes can be used.

```
public void evaluate(MbMessageAssembly assembly) throws MbException {
  MbOutputTerminal out = getOutputTerminal("out");
  MbOutputTerminal alt = getOutputTerminal("alternate");

  MbMessage message = assembly.getMessage();
  int routeTo = 0;
  try {
    // ------------------------------------------------------
```

```
    // Add user code below
    String routingValue = (String)message.evaluateXPath("string(/csv/header/route_to)");
    routeTo = Integer.parseInt(routingValue);


    // End of user code
    // -----------------------------------------------------------
  } catch (java.lang.NumberFormatException e ) {
    // Example Exception handling
    MbUserException mbue = new MbUserException(this, "evaluate()", "","", e.toString(), null);
    throw mbue;
  }

  // The following should only be changed
  // if not propagating message to the 'out' terminal
  if ( routeTo > 0 )
    out.propagate(assembly);
  else
    alt.propagate(assembly);
}
```

Logging errors with the JavaCompute node
The MbService class contains a number of static methods for writing to the Administration log or syslog. Define message catalogs by using Java resource bundles to store the message text.

About this task

Three levels of severity are supported:

- Information
- Warning
- Error

Using XSL Transform

Use the XSLTransform node to transform an XML message to another form of message, according to the rules provided by an XSL (Extensible Stylesheet Language) style sheet.

About this task

Use the XSLTransform node to transform an input XML message into another format using XSLT style sheets and to set the message domain, message model, message type, and message format for the generated message. It is imperative that the data can be parsed into a XML message. The style sheet, using the rules that are defined within it, can perform the following actions:

- Sort the data
- Select data elements to include or exclude based on some criteria
- Transform the data into another format

The Xalan-Java transformation engine (Apache Xalan-java XSLT processor) is used as the underlying transformation engine. For more information about XML Transformations, the W3C specification of the syntax, and semantics of the XSL Transformations language for transforming XML documents into other XML documents, see W3C XSL Transformations.

You can deploy style sheets and XML files to integration servers, to help with style sheet and XML file maintenance.

You can specify the location of the style sheet to apply to this transformation in three ways:

- Use the content of the XML data within the message itself to transform the message according to a style sheet that the message itself defines.
- Set a value in the `LocalEnvironment` folder. You must set this value in a node that precedes the XSLTransform node (for example, a Compute node). You can therefore use various inputs to determine which style sheet to use for this message, such as the content of the message data, or a value in a database.

- Use node properties to ensure that the transformation that is defined by this single style sheet is applied to every message that is processed by this node.

An XSLT (Extensible Stylesheet Language for Transformations) compiler is used for the transformation if the style sheet is not embedded within the message, and the node cache level (node property Stylesheet Cache Level) is greater than zero. If the XSLT is cached, the performance is improved because the XSLT is not parsed every time it is used.

If the prologue of the input message body contains an XML encoding declaration, the XSLTransform node ignores the encoding, and always uses the CodedCharSetId in the message property folder to decode the message.

The XSLT capability that is provided by the XSLTransform node requires XML processing APIs that are included in Xalan-Java and Xerces JAR files. The XSLTransform node provides Xalan-Java and Xerces JAR files that work correctly with the node. The Java JRE also includes Xalan-Java and Xerces JAR files, but you might experience unpredictable results when these Java XML processing methods are invoked by using an external Java method from a style sheet. Therefore, the calling of Java methods from a style sheet that directly or indirectly reference Java JRE XML processing methods is unsupported.

To find out more about the XSLTransform node and how to configure it, refer to the following topics:

- node
- Using local environment variables to set properties
- Deployed and non-deployed style sheets

Using .NET

You can use .NET applications on Windows integration nodes to create, route, and transform messages by using the .NETCompute and .NETInput nodes.

About this task

.NET nodes use any Common Language Runtime (CLR) compliant language to create or modify the message, such as C#, Visual Basic (VB), F#, and C++/CLI (Common Language Infrastructure).

You configure a .NET node with a .NET assembly that contains the code of the node. The code consists of a class that is derived from the abstract class that is provided in the `IBM.Broker.Plugin.dll` assembly, for example, NBComputeNode.

The .NET assembly is run inside a .NET application domain. For more information, read “.NET application domains” on page 1791.

This section describes the following tasks:

.NET application domains

A .NET application domain is a run time container for .NET assemblies and associated resources used by the .NET code in your message flows.

You can call .NET assemblies in your message flows from the .NETCompute node, .NETInput node or from an ESQL procedure. You can also include .NET assemblies as dependencies of a library. These assemblies run in a .NET application domain and can be packaged in a BAR file. Application domains are shown in the navigator view as peers of applications and libraries.

A named .NET application domain can be associated with IBM App Connect Enterprise applications. Assemblies, or associated resources, contained in the .NET application domain are deployed along with the referencing message flow or application. You can associate more than one application with the same .NET application domain. This means that data and other resources can be shared at run time between multiple applications.

Implicit application domains

Creating a .NET application domain in the IBM App Connect Enterprise Toolkit allows you to package your .NET assemblies and associated resources as desired to run on the CLR (Common Language Runtime). However, you do not have to explicitly create a .NET application domain in the IBM App Connect Enterprise Toolkit.

If you do not explicitly create a .NET application domain, and do not set a value for the **AppDomain** property on the .NET node, .NET assemblies can also be contained in an implicit .NET application domain. If you do not set the **AppDomain** node property, an implicit application domain is shown in the navigator when a message flow in your workspace calls a .NET assembly, or when a library in your workspace includes .NET assemblies. At run time the implicit application domain is automatically created by the integration node and takes the name of the referencing application, or the name of the integration server if the message flow is not contained in an application.

This means that the name of the application domain at run time is dependent on the name of the owning application or integration server, and changes if the application or integration server is renamed. Additionally, if an assembly is included in one library that is referenced by two different applications, the application domain under which that assembly runs has a different name dependent on the owning application.

Consider the following example: A library MyLib includes a .NET assembly and is referenced from application App1. When App1 is deployed, a copy of the library is made and the .NET assembly runs in a .NET application domain that takes the name of the application: App1. However, If MyLib is also referenced from application App2, a separate copy of the library is made, and the application domain in which the second copy of the assembly runs takes the name App2.

Deploying .NET code

When you deploy a message flow or application that uses .NET assemblies, there are two ways of working with the assemblies. These typically correspond with the application development, and test or production stages of the software development life cycle:

- You can choose to not package the .NET application domain in the BAR file. In this case, the .NET assembly is loaded directly from your integration node file system. This means that you do not have to redeploy the message flow when the assembly is rebuilt on the file system, because the integration node reloads the application domain with the rebuilt assembly. However, the assembly must always be available on the integration node file system. Use this method for application development systems.
- You can package the .NET application domain in the BAR file. This allows greater portability of message flows on different integration nodes and servers, but means that you must redeploy the BAR file if the .NET assembly is rebuilt. Use this method for test and production systems.

Creating a .NET application domain
Create a .NET application domain as a run time container for .NET assemblies used in your message flows. You can associate .NET application domains with one or more applications as a dependent resource.

Before you begin

Read the concept information ".NET application domains" on page 1791.

About this task

Use this process to create a .NET application domain, in which you can contain .NET assemblies and other resources that will run within that execution scope at run time. You can associate a .NET application domain with one or more applications as a dependent resource.

To create a .NET application domain, complete the following steps.

Procedure

1. To open the New .NET Project wizard, click **File** > **New** > **.NET Project**.

   Alternatively, in the Application Development view, right-click within the navigator and click **New** > **.NET Project**.
2. Specify the name of the application domain to create.
3. Click **Finish** to create an empty application domain, or click **Next** to import .NET assemblies or other resources into the application domain from the file system.

   You also can drag and drop assemblies from the file system onto the application domain after closing the wizard.
4. To import .NET assemblies into the application domain, select a root directory and choose the assemblies or other resources that you want to import.
5. Click **Finish**.

Results

The new application domain is displayed in the Application Development view, as a peer of applications and libraries.

Importing resources into a .NET application domain

Import .NET assemblies and other resources into a .NET application domain.

Before you begin

Create a .NET application domain. For more information, see "Creating a .NET application domain" on page 1792.

About this task

When you import a .NET assembly into an application domain, the import stores a link to the original location of the file on the file system. Changes made to the assembly on the file system are picked up when you refresh the application domain object.

The import additionally makes a copy of the imported file in the application domain, and so you can deploy your .NET application from your workspace even if you remove the original source file from the file system.

Deploying .NET assemblies

.NET assemblies can be included in a BAR file, or loaded directly from your integration node file system.

About this task

Include the .NET assembly in the BAR file by packaging it in a .NET application domain. The application domain is packaged as a `.appdomainzip` at the root level of the BAR file. The .NET assembly is deployed with any other resources in the application domain.

If the .NET application domain is associated with an application, you can deploy the referencing application. The .NET assemblies in the application domain are deployed with the application. Alternatively, you can deploy the .NET application domain separately.

Deploy the application domain by right-clicking on the object in the Application Development view, or by dragging and dropping the application domain onto an integration server in the Integration Explorer view.

If the .NET assembly is not included in the BAR file, and is loaded directly from the integration node file system, the assembly is found in one of the following ways:

- The absolute directory path specified on a .NET node **Assembly name** property, or specified on an ESQL function or procedure.
- An override to the absolute directory path in a BAR file for the .NET node.
- A DotNet Application Domain policy that overrides the absolute directory path.
- Using advanced properties to search the global assembly cache (GAC).

Identifying the .NET assembly at run time

The rules that govern how the .NET assembly is loaded from the integration node file system if it is not packaged in the BAR file.

About this task

The .NET configuration for an integration server is controlled by properties on the .NET node or the ESQL Procedure signature. Optionally, a DotNet Application Domain policy can also be defined for further configuration options. A message flow can invoke assemblies that run within an integration server that uses the Microsoft Common Language Runtime (CLR). Flow developers can control the .NET framework assemblies that are loaded, and the .NET application domains which are used. Assemblies are invoked either from a .NET node in a message flow, or by calling an ESQL procedure.

Application domains

About this task

The .NET framework separates application assemblies to run in a particular application domain that provides a scope for the resources for an application. In IBM App Connect Enterprise, the integration server operating system process (DataFlowEngine) hosts the CLR. Within this process, .NET assemblies can be executed within multiple application domains if required.

- The **AppDomain name** property of a .NET node specifies the application domain in which the assembly is to be loaded. If a DotNet Application Domain policy is defined with the same name as the **AppDomain name** property of the node, the policy properties take precedence when the assembly is loaded. Such properties include the **Application Base directory** property.
- When an ESQL procedure invokes a .NET assembly, the signature of the procedure can specify the application domain in which the assembly must be loaded. If a DotNet Application Domain policy is defined with the name that is specified for the application domain in the signature of the ESQL procedure, the policy properties (for example the **Application Base directory**) take precedence when the assembly is loaded.

If no application domain is specified by the preceding methods, the application domain in which the assembly is loaded is determined by:

- The name of the application, if deployed within an IBM App Connect Enterprise application
- The name of the integration server, if deployed in a flow which is not defined within an IBM App Connect Enterprise application

If the application domain is determined by the name of the application or the name of the integration server, and a DotNet Application Domain policy is defined with the same name, the policy properties (for example, the **Application Base directory**) take precedence when the assembly is loaded. If the application name or integration server name includes space characters, these characters are ignored when the application name or integration server name is compared to the name of the policy.

Application base directory

About this task

To understand which copy of an assembly will be loaded, first determine the Application Base directory.

1. If the `Application domain` property refers to a policy, the **Application base** property of the policy is used. Use this approach in production environments.

   Note: The policy properties, including **Application base**, always take precedence even if you have deployed a .NET application domain that contains the assemblies to the run time. In this situation, to ensure that the .NET assembly is loaded from the `.appdomainzip` and not from the Application Base directory, ensure that the **Application base** property of the policy is blank.
2. If the Assembly name property of the .NET node (or equivalent part of the ESQL procedure signature) specifies an absolute location in the file system, this full directory path is used as the `Application Base directory`.
3. If the Assembly name property of the .NET node (or equivalent part of the ESQL procedure signature) specifies just the name of the .dll file with no path, the integration node uses a subdirectory of the runtime integration node's work path for the Application Base directory. This directory might change; consider using one of the alternative approaches.

Assembly loading and the Global Assembly Cache

About this task

The integration node expects to find the required .NET assembly in the Application Base directory. However, you can place assemblies in a machine-wide cache, which is known as the Global Assembly Cache (GAC). To place an assembly into the GAC, it must be *Strong-Named*: when it is compiled, the assembly is given the properties of a version, a culture, and a public key token. When configuring the .NET

node or an invocation from ESQL, you can specify the version, culture, and public key token of the *Strong-Named* assembly to be used.

You have to restart the integration server if you want to reload or update a *Strong-Named* assembly.

Creating and transforming messages using a .NETCompute node

You can use the .NETCompute node to create, route, and copy messages, and to create and manipulate message elements.

About this task

This section describes the following tasks:

Creating a message with a .NETCompute node

Write appropriate code in the UserCode region of a CreateNode class.

Procedure

1. In Microsoft Visual Studio, add an instance of the **Class to create a Message Broker message** to your project.
2. Add the appropriate code to the UserCode region of the class that you created.
   The following example creates a simple message:

```
outputRoot.CreateLastChildUsingNewParser(NBParsers.XMLNSC.ParserName);
outputRoot["XMLNSC"].CreateFirstChild(null, "Message");
outputRoot["XMLNSC"]["Message"].CreateFirstChild(null, "Element").SetValue("Some Data");
```

3. Build your project, and then associate the `.dll` file with a .NETCompute node.
   If the example code is associated with a .NETCompute node between an MQInput node and an MQInput node, the following message is created:

```
<Message><Element>Some Data</Element></Message>
```

Copying a message with a .NETCompute node

Copy an existing message using the .NETCompute node.

About this task

Transformation nodes, such as the .NETCompute node, can modify the logical tree structure that is passed through a message flow from one node to the next. This tree structure, or *message assembly*, contains four trees to represent the message, the environment, the local environment, and the exception list. To send data from the input terminal of the .NETCompute node to the output terminal, the class associated with the node must contain an Evaluate method that either propagates the input message assembly, or creates an output message assembly. To copy a message from the input terminal to the output terminal, propagate the whole input message assembly. The template code provided for you when

you create a FilterNode class in C# propagates the message. As you can see in the following code, the inputAssembly is passed to the Propagate method:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");

            NBMessage inputMessage = inputAssembly.Message;
            NBElement root = inputMessage.RootElement;

            #region UserCode
            // Add user code in this region to filter the message
            #endregion UserCode

            // Change the following if not propagating message to the 'Out' terminal
            outTerminal.Propagate(inputAssembly);
        }
```

If you plan to copy the message and then also update it in the .NETCompute node, it is better to use the CreateNode template. This template creates an output assembly. You copy the input assembly to this output assembly by adding just one line of code to the UserCode region:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("out");

            NBMessage inputMessage = inputAssembly.Message;

            // Create a new empty message, ensuring it is disposed after use
            using (NBMessage outputMessage = new NBMessage())
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
 outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                // Optionally copy message headers, remove if not needed
                CopyMessageHeaders(inputRoot, outputRoot);

                #region UserCode
                // Add user code in this region to create a new output message
                outputRoot.AddLastChild(inputRoot.LastChild);
                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
```

Routing a message using a .NETCompute node
Route a message by using the .NETCompute node as a filter node.

Before you begin

Add a .NETCompute node to your message flow.

About this task

By default, the output message assembly is propagated to the Out terminal after the evaluate method in the .NET code has been processed. However, the .NETCompute node supports dynamic terminals. You can create extra terminals, and use the .NETCompute as a filter node by propagating a message to the appropriate terminal, based on the message content.

The following snippet of C# code shows how you might filter a message, depending on the content of an element in the message:

```
            #region UserCode
            // Add user code in this region to filter the message

            if (root[NBParsers.XMLNSC.ParserName].LastChild.Name.Equals("LoyaltyProgram"))
            {
```

```
            outTerminal.Propagate(assembly);
        }

        if (root[NBParsers.XMLNSC.ParserName].LastChild.Name.Equals("SaleEnvelope"))
        {
            altTerminal.Propagate(assembly);
        }
        else
        {
            failureTerminal.Propagate(assembly);
        }
        #endregion UserCode
```

Setting and moving message elements using a .NETCompute node
You can transform elements in the message as it passes through a .NETCompute node in the message flow.

About this task

The following sections show the methods that you can use to set, move, and remove elements:

- "Setting information about an element" on page 1797
- "Moving elements" on page 1797
- "Removing elements" on page 1798

Setting information about an element

About this task

Use these properties and methods to set information about the referenced element:

Name
This property sets the name of the current element

ValueType
This property sets the specific type of the current element.

Namespace
This property sets the namespace URI of the current element

SetValue
This method sets the value of the current element

Moving elements

About this task

Use the following methods to copy or detach an element from a message tree:

Detach
This method detaches the current element from the tree.

DetachAllChildren
This method detaches all children of the current element from the tree.

Use one of the following methods to attach an element or subtree that you have copied on to another tree:

AddBefore
This method adds an element before the referenced element.

AddAfter
This method adds an element after the referenced element.

AddFirst
This method adds an element as the first child of the referenced element.

AddLast
This method adds an element as the last child of the referenced element.

Removing elements

About this task

Use these properties and methods to remove elements from the message tree:

Delete
This method deletes the referenced element from the tree.

DeleteAllChildren
This method deletes all children of the referenced element from the tree.

Creating elements using a .NETCompute node
The NBElement class represents a single parsed element in the message tree. The class provides four different creation methods that you can use to create a new element in the message tree.

About this task

- CreateFirstChild
- CreateLastChild
- CreateBefore
- CreateAfter

The method names indicate where the element is to be created in the hierarchy in relation to the NBElement on which they are called. The following C# Evaluate method contains example code for a ..NETCompute node:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");
            NBMessage inputMessage = inputAssembly.Message;

            // Create a new empty message, ensuring it is disposed after use
            using (NBMessage outputMessage = new NBMessage())
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                // Optionally copy message headers, remove if not needed
                CopyMessageHeaders(inputRoot, outputRoot);

                #region UserCode
                // Add user code in this region to create a new output message
                NBElement msg =
outputRoot.CreateLastChildUsingNewParser(NBParsers.XMLNSC.ParserName).CreateFirstChild(null,"Message");
                NBElement el4 = msg.CreateLastChild("Element4");
                NBElement el1 = msg.CreateFirstChild("Element1");
                el1.SetValue("Data Value for Element1");
                el4.SetValue("Data Value for Element4");
                el1.CreateAfter("Element2").SetValue("Data Value for Element2");
                el4.CreateBefore("Element3").SetValue("Data Value for Element3");

                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
```

When the node is wired to a suitable output node (such as an MQOutput node), the code produces an XML message that looks like this:

```
<Message>
     <Element1>Data Value for Elememt1</Element1>
     <Element2>Data Value for Elememt2</Element2>
     <Element3>Data Value for Elememt3</Element3>
     <Element4>Data Value for Elememt4</Element4>
</Message>
```

Writing code for a .NETCompute node
Create code for message flows containing a .NETCompute node by double-clicking the node to start Microsoft Visual Studio.

Before you begin

- Install Microsoft Visual Studio 2010 or 2012.
- Create a message flow that contains a .NETCompute node.

Tip: If the IBM App Connect Enterprise Toolkit is installed after Microsoft Visual Studio, the integration node Project templates will be automatically installed ready for you to use. However, if the IBM App Connect Enterprise Toolkit is installed first, you must manually install the Microsoft Visual Studio templates. This installation can be achieved by running the file `IBM.Broker.DotNet.vsix` and stepping through the wizard (accepting the license file as part of the process). If the default installation location was used for IBM App Connect Enterprise Toolkit, the file can be found at the location `C:\Program Files\IBM\ACE\12.0.`*n*`.0\tools\iibt`.

About this task

You write and compile your code in Microsoft Visual Studio. The following IBM App Connect Enterprise templates are provided to get you started in C#, Visual F#, and Visual Basic:

- **Create an IBM Integration message**. This class also provides a method to copy message headers from an incoming message.
- **Filter an IBM Integration message**.
- **Modify an IBM Integration message**.

Procedure

1. Double-click the .NETCompute node.
2. In Microsoft Visual Studio 2010 or 2012, navigate to your project, and add the appropriate class from the **IBM Integration** templates.
3. Expand the `UserCode` region, and write your code.

   Content assist is available in this region. Press `Ctrl+Space` to invoke it.
4. Save the class and build the solution, making a note of the path to the assembled file.

What to do next

Associate your assembly with the .NETCompute node.

Associating code with a .NETCompute node
To associate your compiled code assembly with a .NETCompute node, complete the **`Assembly name`** and **`Class name`** properties on the node.

About this task

Tip: Your message flows can call .NET assemblies. You write and maintain your .NET code in Microsoft Visual Studio. You develop your message flows in the IBM App Connect Enterprise Toolkit, which runs in an Eclipse framework. It is suggested that you keep the assemblies in the Visual Studio framework; do not import them into the IBM App Connect Enterprise Toolkit.

Tip: You can also drag an existing .NET assembly or .NETCompute node onto the canvas, and then follow the prompts to create this association. The assembly must contain an implementation of a .NETCompute node.

Procedure

1. Create a message flow that contains a .NETCompute node.
   For example, create a message flow containing an MQInput node, a .NETCompute node, and an MQOutput node.
2. On the **Basic** tab of the .NETCompute node, set the **`Assembly name`** property to point to the assembly that contains your compiled code.

   You can either type the full path name of the file into the field, or select **Browse** to navigate to `.exe` and `.dll` files stored on the file system.
3. Optional: Specify the name of the class within the assembly.
4. Save the message flow.

Traversing the element tree by using a .NETCompute node
Use .NETCompute node NBElement properties and methods (in C# for example) to access element trees.

About this task

Use the following statements to traverse a message tree from an NBElement:

Parent
This property returns the parent of the current element.

PreviousSibling
This property returns the previous sibling of the current element.

NextSibling
This property returns the next sibling of the current element.

Children()
This method returns all the child elements of the current element as an array of NBElements.

FirstChild
This property returns the first child of the current element.

LastChild
This property returns the last child of the current element.

Iterating over elements by using a .NETCompute node
Use the **`Where`** method to provide a sequential iteration over elements in the message tree.

About this task

You can use the resulting array in a foreach() statement to iterate over the children of an NBElement. The foreach() loop allows a developer to act on each NBElement in turn. The example below shows how this can be used to create a similar shape output message to the input message. Consider a simple XML input message:

```
<MessageEnvelope>
 <Element>1</Element>
 <Element>2</Element>
 <Element>3</Element>
</MessageEnvelope>
```

The following C# code in a .NETCompute node iterates over the elements:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("out");

            NBMessage inputMessage = inputAssembly.Message;

            // Create a new empty message, ensuring it is disposed after use
            using (NBMessage outputMessage = new NBMessage())
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
 outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
```

```
                NBElement outputRoot = outputMessage.RootElement;

                // Optionally copy message headers, remove if not needed
                CopyMessageHeaders(inputRoot, outputRoot);

                #region UserCode
                // Add user code in this region to create a new output message
                outputRoot.CreateLastChildUsingNewParser("XMLNSC");
                outputRoot["XMLNSC"].CreateLastChild("OutputMessageEnvelope");
                string notargetnamespace = "";
                NBElement env = inputRoot["XMLNSC"]["MessageEnvelope"];
                var elementlist = env.Where(t => t.Name == "Element");
                foreach (NBElement element in elementlist)
                {
                    int currentvalue = (Int32)element;
                    NBElement outelement = outputRoot["XMLNSC"]
["OutputMessageEnvelope"].CreateLastChild(notargetnamespace, "OutElement");
                    outelement.SetValue("After adding 1 to input value we get " + (currentvalue
 + 1).ToString());
                }
                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
```

The output is an XML message:

```
<OutputMessageEnvelope>
 <OutElement>After adding 1 to input value we get value 2</OutElement>
 <OutElement>After adding 1 to input value we get value 3</OutElement>
 <OutElement>After adding 1 to input value we get value 4</OutElement>
</OutputMessageEnvelope>
```

Accessing other parts of the message tree using a .NETCompute node
Use the .NETCompute node to access headers, integration node properties, user-defined properties, the local environment, and the global environment.

About this task

The message tree is passed to a .NETCompute node as an argument of the evaluate method. The argument is an NBMessageAssembly object. The message assembly contains four message objects:

- Parsed message
- LocalEnvironment
- GlobalEnvironment
- ExceptionList

The following topics describe how to access parts of the message tree:

- “Accessing headers with a .NETCompute node” on page 1801
- “Updating the local environment with a .NETCompute node” on page 1802
- “Updating the global environment by using a .NETCompute node” on page 1802
- “Accessing user-defined properties from a .NETCompute node” on page 1803

Accessing headers with a .NETCompute node
Use a .NETCompute node to access headers in the message assembly.

About this task

Two of the most common headers encountered in messaging scenarios are the MQMD and MQRFH2. If an input node receives an input message that includes message headers that the input node recognizes, the node invokes the correct parser for each header. Parsers are supplied for most IBM MQ headers. This topic provides guidance for accessing the information in the MQMD and MQRFH2 headers that you can follow when accessing other headers that are present in your messages.

For more information about the contents of these and other IBM MQ headers for which IBM App Connect Enterprise provides a parser, see Element definitions for message parsers.

The following C# code shows how to add an MQMD header and an MQRFH2 header to your message using a .NETCompute node:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("out");

            NBMessage inputMessage = inputAssembly.Message;

            // Create a new empty message, ensuring it is disposed after use
            using (NBMessage outputMessage = new NBMessage())
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                #region UserCode
                // Add user code in this region to create a new output message
                NBElement MQMD =
outputRoot.CreateLastChildUsingNewParser(NBParsers.NBHeaderParsers.MQMD.ParserName);

                NBElement MQRFH2 =
outputRoot.CreateLastChildUsingNewParser(NBParsers.NBHeaderParsers.MQRFH2.ParserName);
                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
```

Updating the local environment with a .NETCompute node
The local environment tree is part of the logical message tree where you can store information while the message flow processes the message.

About this task

In the following example, the C# Evaluate method, which is executed by a .NETCompute node, creates an output message assembly based on the input message assembly. The local environment that enters the .NETCompute node is copied and then updated with the name of a queue which can be used as a DestinationList to dynamically control the behavior of a later MQOutput node.

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");
            // This code creates an output Assembly based on the input Assembly
            // The Local Environment is copied and editted to provide a dynamic override for the
MQOutput node
            NBMessage inputLocalEnvironment = inputAssembly.LocalEnvironment;
            NBMessage outputLocalEnvironment = new NBMessage(inputLocalEnvironment);
            NBElement mqLE = outputLocalEnvironment.RootElement.CreateFirstChild(null,
"Destination").CreateFirstChild(null, "MQ");
            mqLE.CreateFirstChild(null, "DestinationData").CreateFirstChild(null, "queueName",
"DOTNET.OUT");
            NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
inputAssembly.Message, outputLocalEnvironment, inputAssembly.ExceptionList);

            outTerminal.Propagate(outAssembly);
        }
```

Updating the global environment by using a .NETCompute node
Change the global environment by using code in your .NETCompute node.

About this task

The global environment tree is always created when the logical tree is created for an input message. However, a message flow does not populate it or use its contents. You can use this tree for your own purposes, for example to pass information from one node to another. You can use the whole tree as a scratchpad or working area.

The Global Environment can be altered at any point during the message flow; therefore you do not make a copy of it to alter. The following C# code shows how to change the global environment in a .NETCompute node:

```
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");
            NBMessage inputMessage = inputAssembly.Message;
            // Create a new empty message, ensuring it is disposed after use
            using (NBMessage outputMessage = new NBMessage())
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly,
 outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                // Optionally copy message headers, remove if not needed
                CopyMessageHeaders(inputRoot, outputRoot);

                #region UserCode
                // Add user code in this region to create a new output message
                NBMessage env = outAssembly.Environment;
                env.RootElement.CreateFirstChild(null, "Status", "Success");

                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
```

Accessing user-defined properties from a .NETCompute node
You can customize a .NETCompute node to access properties that you have associated with the message flow in which the node is included.

About this task

To access these properties from a .NETCompute node, use the GetUserDefinedProperty or GetUserDefinedPropertyAsString methods. The following sample C# code below shows how the methods can be used:

```
                #region UserCode
                // Add user code in this region to create a new output message
                int intProperty = (int)this.GetUserDefinedProperty("intProperty");
                string stringProperty = this.GetUserDefinedPropertyAsString("stringProperty");
                NBElement msg =
 outputRoot.CreateLastChildUsingNewParser(NBParsers.XMLNSC.ParserName).CreateFirstChild(null,
 "Message");
                NBElement el2 = msg.CreateFirstChild("Element2");
                NBElement el1 = msg.CreateFirstChild("Element1");
                el1.SetValue("The message flow user defined property named intProperty has the
 value "+intProperty.ToString());
                el2.SetValue("The message flow user defined property named stringProperty has
 the value "+stringProperty);
                #endregion UserCode
```

This code produces the following message (where the integer and string flow properties are set to the values 23 and "Hello World!" respectively):

```
<Message>
 <Element1>The message flow user defined property named intProperty has the value 23</Element1>
 <Element2>The message flow user defined property named stringProperty has the value Hello
 World!</Element2>
</Message>
```

Serializing .NET custom exceptions
To make .NET custom exceptions visible in IBM App Connect Enterprise, you must serialize them.

About this task

To be visible in IBM App Connect Enterprise, you must make .NET custom exceptions serializable, which means converting them into a state that can be persisted. For more information about serialization, see the Microsoft .NET framework documentation.

Creating messages by using a .NETInput node

You can use the .NETInput node to create, route, and copy messages, and to create and manipulate message elements.

About this task

This section describes the following features and tasks associated with the .NETInput node:

- "Writing code for a .NETInput node" on page 1804
- "Associating code with a .NETInput node" on page 1805
- "Accessing .NETInput node properties at run time" on page 1806
- "Adding dynamic output terminals to a .NETInput node" on page 1806
- "Adding local environment entries from a .NETInput node" on page 1807
- ".NETInput node exception handling" on page 1808

Writing code for a .NETInput node
Create code for message flows containing a .NETInput node by double-clicking the node to start Microsoft Visual Studio.

Before you begin

- Install Microsoft Visual Studio 2010 or 2012.
- Create a message flow that contains a .NETInput node.

Tip: If the IBM App Connect Enterprise Toolkit is installed after Microsoft Visual Studio, the integration node Project templates will be automatically installed ready for you to use. However, if the IBM App Connect Enterprise Toolkit is installed first, you must manually install the Microsoft Visual Studio templates. This installation can be achieved by running the file `IBM.Broker.DotNet.vsix` and stepping through the wizard, accepting the license file as part of the process. If the default installation location was used for IBM App Connect Enterprise Toolkit, the file can be found at the location `C:\Program Files \IBM\ACE\12.0.n.0\tools\iibt`.

About this task

You write and compile your code in Microsoft Visual Studio. The following IBM App Connect Enterprise templates are provided for C#, Visual F#, and Visual Basic:

- **Create a polling-based IBM Integration input node**.
- **Create an event-driven IBM Integration input node**.

Procedure

1. Double-click the .NETInput node. Alternatively, right-click the node and select **Open Microsoft Visual Studio**.

   If there is a Microsoft Visual Studio project or solution name on the **Visual Studio** panel, the project or solution automatically loads when Microsoft Visual Studio opens.

2. In Microsoft Visual Studio, select **New Project**.

   You can then select an **IBM Integration project** from beneath the language of your choice; C#, F# and Visual Basic.
3. Select a project from the list. For the .NETInput node, you can choose:

   - **Create a polling-based IBM Integration input node**
   - **Create an event-driven IBM Integration input node**

   The project then generates skeleton code that matches the selected type. This skeleton code then displays in the corresponding .NET language editor, for example; C#.
4. Expand the `UserCode` region, and write your code.

   Content assist is available in this region. Press `Ctrl+Space` to invoke it.
5. Optional: Use the methods available for your .NETInput node.

   See "Methods for a .NETInput node" on page 1808.
6. Save the class and build the solution, making a note of the path to the assembled file.

What to do next

- You might want to debug your .NET code.

  To debug a.NETInput node that is running inside the integration server, place the `.pdb` file for the assembly that implements the node in the same directory as the assembly for the node, and attach Microsoft Visual Studio to the `DataFlowEngine.exe` process. You can then place breakpoints in your code and examine variables, for example; when the breakpoint is hit.
- Associate your code with the .NETInput node.

Associating code with a .NETInput node
To associate your compiled code assembly with a .NETInput node, complete the **`Assembly name`** and **`Class name`** properties on the node.

About this task

Tip: Your message flows can call .NET assemblies. You write and maintain your .NET code in Microsoft Visual Studio. You develop your message flows in the IBM App Connect Enterprise Toolkit, which runs in an Eclipse framework. It is suggested that you keep the assemblies in the Microsoft Visual Studio framework; do not import them into the IBM App Connect Enterprise Toolkit.

Tip: You can also drag an existing .NET assembly or .NETInput node onto the canvas, and then follow the prompts to create this association. The assembly must contain an implementation of a .NETInput node.

Procedure

1. Create a message flow that contains a .NETInput node.
   For example, create a message flow that contains an MQInput node, a .NETInput node, and an MQOutput node.
2. On the **Basic** tab of the .NET node, set the **`Assembly name`** property to point to the assembly that contains your compiled code.

   You can either type the full path name of the file into the field, or select **Browse** to navigate to `.exe` and `.dll` files that are stored on the file system.
3. Optional: Specify the name of the class within the assembly.
4. Save the message flow.

Accessing .NETInput node properties at run time
When you add a .NETInput node to a message flow, you can use the **User Defined Properties** tab of the node to specify properties that can be accessed by the .NET connector code, which implements the logic of the input node at run time.

About this task

The .NET framework provides a class in the namespace `System.Collections.Generic` named `Dictionary<TKey, TValue>`, which represents a collection of keys and their associated values. When you write code for a .NETInput node, the templates that are provided in Microsoft Visual Studio helps you initialize a new instance of the `EventInputConnector` or the `PollingInputConnector` class. The template code includes the definition of the subclass for the `EventInputConnector` or `PollingInputConnector` class, which contains a **`properties`** parameter, which is of the data type `Dictionary<string,string>`. This parameter carries user-defined properties for the node, and any flow-level user-defined properties.

The individual property names are case-sensitive. For example, you can define a flow-level user-defined property named `property1`, which would be independent of a .NETInput node **User Defined Property** named `Property1`. If you specify a flow-level user-defined property and a .NETInput node user-defined property with the same name and case, then the value of the .NETInput node property takes precedence.

The following code snippet demonstrates a simple example:

```
public PollingInputConnector(NBConnectorFactory connectorFactory, string name,
 Dictionary<string, string> properties) : base(connectorFactory, name, properties)
{
// Check the Dictionary of user defined properties defined on the node.
// If there is a property named queueName then take its value for use later:
if (properties.ContainsKey("queueName"))
{
queueName = properties["queueName"];
}
}
```

Adding dynamic output terminals to a .NETInput node
You can add, rename, and remove dynamic output terminals on a .NETInput in the Message Flow editor.

Before you begin

- Add a .NETInput node to your message flow that supports dynamic terminals. For more information, see “Adding a message flow node” on page 618, “Message flow node terminals” on page 497, and “Writing code for a .NETInput node” on page 1804.

About this task

The .NETInput node can support dynamic output terminals. When you have added the .NETInput node to the flow editor, you can add, remove, or rename dynamic terminals.

Procedure

- **Adding a dynamic terminal**
  a) Right-click the node and click **Add Output Terminal**.
  b) Enter a name for the new terminal and click **OK**.
  The name must be unique for the terminal type. For example, if an output terminal called *Out* already exists, you cannot create a dynamic output terminal with the name *Out*.

  The new terminal is displayed on the node. If a node has five or more terminals, they are displayed as a terminal group: 

  To connect a particular output terminal, click the terminal group to open the Terminal Selection dialog box, or right-click the node and select **Create Connection**.
- **Renaming a dynamic terminal**
  a) Right-click the node and click **Rename Output Terminal**.
  These options are available only if you have added one or more appropriate terminals to the node.
  b) Select from the list the name of the terminal that you want to change.
  Only dynamic terminals are listed because you cannot change the name of a static terminal.
  c) Enter a new name for the terminal and click **OK**.
  Do not rename a dynamic terminal if one of the node properties is configured to use that name.
- **Removing a dynamic terminal**
  a) Right-click the node and click **Remove Output Terminal**.
  These options are available only if you have added one or more appropriate terminals to this node.
  b) Select from the list the name of the terminal that you want to remove and click **OK**.
  Only dynamic terminals are listed because you cannot remove a static terminal. Do not remove a dynamic terminal if one of the node properties is configured to use that terminal.

What to do next

When you have added dynamic output terminals to the .NETInput node, connect them to other nodes in the message flow. For more information, see “Connecting message flow nodes” on page 636.

Adding local environment entries from a .NETInput node
Add information into the local environment by using the connector code for a .NETInput node.

About this task

When developing for a .NETInput node, the content of the local environment can be populated from within the connector code for the .NETInput node as each message is created. The updated local environment allows metadata about the specific `Event` or `PollingResult` to be stored for use by later nodes in the flow.

The data is put in a section of the local environment that is called `DotNet` by default. Under `DotNet` in the folder structure is an `Input` folder. You can create further folders and elements under the `Input` folder.

You can change the top-level `DotNet` folder name by setting the **Name** property of the factory from within the connectors `Constructor()`, `Initialize()` or `Start()` methods.

You cannot change the `Input` folder name.

To create folders and elements below the `Input` folder, override the `BuildProperties` method on a subclass of `NBPollingResult` or `NBByteArrayPollingResult`. In this method, return a Dictionary that is populated with keys and values that become **Name** or **NameValue** elements under the `Input`

folder. If the key has multiple parts that are separated by / characters, then a local environment tree is created, as shown by the following code examples.

This code snippet is from a user-defined `NBPollingResult` subclass:

```
public override Dictionary<string, string> BuildProperties()
    {
      var result = new Dictionary<string, string>();
      result.Add("a/b/c/d", "Hello");
      result.Add("a/e/f", "World!");
      return result;
    }
```

The previous code results in a Local Environment tree that looks like this:

```
  (0x01000000:Name):DotNet = (
    (0x01000000:Name):Input = (
      (0x01000000:Name):a = (
        (0x01000000:Name):b = (
          (0x01000000:Name):c = (
            (0x03000000:NameValue):d = 'Hello' (CHARACTER)
          )
        )
        (0x01000000:Name):e = (
          (0x03000000:NameValue):f = 'World!' (CHARACTER)
        )
      )
    )
  )
```

The next code snippet shows how to change the name of the top-level `DotNet` folder to the example name `MyConnectorFactory`:

```
    public override void Initialize()
    {
      NBConnectorFactory factory = ConnectorFactory;
      factory.Name = "MyConnectorFactory";
      //Other code here.    }
```

.NETInput node exception handling
The .NETInput node can send exceptions to the Windows event log.

About this task

If the `ValidateData` method input record throws an exception, the exception is written to the **Failure** terminal, if it is wired. If the **Failure** terminal is not wired, then the exception is written to the Windows event log. Exceptions that are thrown in a .NETInput node before `ValidateData` is called (such as the `ReadData` method or the user Delegate) are written to the Windows event log, and an error message is not sent to the **Failure** terminal.

Unhandled exceptions that are thrown by nodes that are wired downstream of the output terminal of the .NETInput node are returned to the .NETInput node and directed to its **Catch** terminal, if wired. If the **Catch** terminal is not wired, the exception is written to the Windows event log.

Methods for a .NETInput node
The methods that you can implement in a .NETInput node depends on if you are using polling or event style message flows.

The API for the .NETInput node is called the **Connector API**. This API is in the `IBM.Broker.Plugin.Connector` namespace.

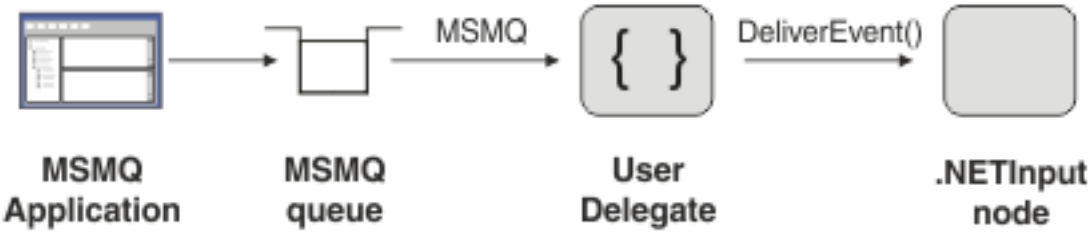
`Initialize`
The `Initialize` method is called when a message flow that contains the .NETInput node is deployed to an integration server. If an exception is thrown from `Initialize`, it rolls back the deployment of the flow.

Start

Use the Start method to create any required connections to the end system from which the .NETInput node retrieves data. This method is called after the Initialize method, when a message flow that contains the .NETInput node is successfully deployed.

If the Start method throws an exception, then it does not influence the result of the deployment of the message flow. Such an exception reports an error to the system log, and the Start method is called again after a short wait.

Event Connectors



If you are using the event-driven style of .NETInput node, then you must register a user-defined delegate with the application that is the source of the events, which drive the .NETInput node. If you use the event-driven style and do not register a delegate in the Start method, then there is no way for the .NETInput node to propagate data to the message flow threads. The User Delegate method is responsible for creating NBEvent objects and passing them to the DeliverEvent method provided on the NBEventConnector class. The DeliverEvent method puts data on an in-memory queue, which the message flow threads access to pass the data in the NBEvent object.

ReadData

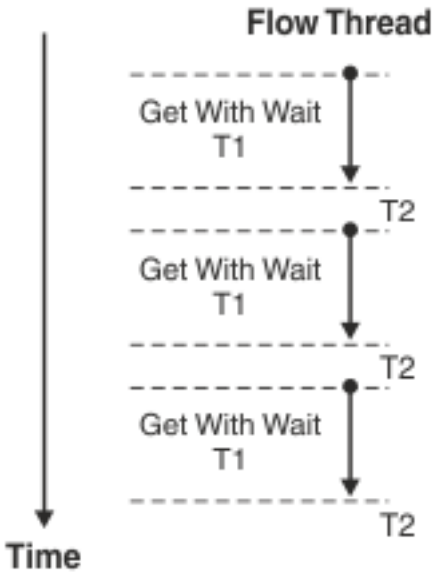
Polling-driven Connectors

If you are using the polling style of .NETInput node, you must implement the ReadData method from the NBPollingConnector base class. Your ReadData method is called by the internal code of the integration node, so a .NETInput node developer implements the method only, and is not responsible for providing code that actually calls the method. Therefore, the .NETInput is not responsible for implementing any recursive polling logic, and so it is not necessary or suitable to code a loop or use a Sleep method as part of ReadData.

The integration node runs the ReadData method automatically after the message flow finishes deploying, when the Start method returns without an exception.

The ReadData method is responsible for taking data from the data source, if available, and returns an instance of the NBPollingResult object. If data is available, the returned NBPollingResult object is either a built-in NBByteArrayPollingResult, which creates the output message directly from the byte array, or a user-defined subclass, from which you can create the message tree manually.

See the following diagram for an example of using polling with the ReadData method.



If no data is available, the node developer returns an `NBTimeoutPollingResult`. When the `ReadData` method has finished waiting for the passed in timeout duration (T1), use the single-argument constructor for `NBTimeoutPollingResult`. If the data source does not allow a timed wait, use the two-argument constructor to return a timeout value back to the integration node. In this case, the integration node waits for this timeout interval (T2) before it calls `ReadData` again. This use of the method allows a developer for the .NETInput node to control the duration of the interval between polls (T2).

For example, you can implement a batch style of operation, requesting that the `ReadData` method is only called once per hour, or once per day.

`Finish`
The `Finish` method is called when the .NETInput node that is using the connector must stop receiving data. When you remove a message flow that contains a .NETInput node that is deployed to an integration server, the `Finish` method is called before the `Terminate` method. If the `Finish` method throws an exception, then the error is reported to the system log, and the connector is placed into stopped state. This method is a good place to close connections or handles, or similar.

`Terminate`
The `Terminate` method is called when a message flow containing the .NETInput node is removed from an integration server. This method always provides a way for your code to end cleanly if you want to do anything before you remove a deployed message flow that contains a .NETInput node.

Cloning a .NETInput node
You can reproduce parameters and attributes that you customized for a .NETInput node by creating a Cloned node from it.

Before you begin

- To create a Cloned node, you must have administrator privileges.
- Add a .NETInput node to a message flow and configure the node.

About this task

The Cloned node has no special properties of its own, and cannot be created from the palette drawer. You can create it only from an existing, configured .NETInput node.

To create a Cloned node, complete the following steps.

Procedure

1. Open the message flow that contains your configured .NETInput node.
2. Right-click the .NETInput node and select **Create Cloned Node**.
3. Enter a name for the cloned node, and provide tooltip text that is displayed when you hover a mouse over the node on the palette.
4. Import node icons to represent the cloned node on the palette and in the Message Flow editor.
5. Click **Finish**.

Results

Your Cloned node is saved in the **.NET** drawer of the palette with the name that you gave it, and is represented in the IBM App Connect Enterprise Toolkit by your chosen node icon.

What to do next

When you create an instance of a Cloned node, it is configured with the properties and values that you specified for the .NETInput node from which it was cloned. To modify properties or values of a Cloned node, configure them by following the instructions in "Configuring a message flow node" on page 624.

You can configure an instance of a Cloned node when the node is dragged into a graphical data map. To configure the node, drag an assembly file from a file explorer to the node. The properties of the node are displayed in the Properties view.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Calling .NET assemblies from ESQL

Use the CREATE FUNCTION or CREATE PROCEDURE statements in a node that supports ESQL.

About this task

You can call .NET code from any of the nodes that support ESQL; see "ESQL overview" on page 1608.

Tip: To associate .NET code with an ESQL node, drag an existing .NET assembly onto the canvas, or a node that supports ESQL, and then follow the prompts.

Procedure

1. In Microsoft Visual Studio, create and build your project.
2. In the IBM App Connect Enterprise Toolkit, drag one of the nodes that support ESQL onto the message flow.
3. Double-click the node to open the ESQL editor.
4. Define a CREATE FUNCTION or CREATE PROCEDURE statement, specifying the language as either .NET or CLR.
   See CREATE FUNCTION statement and CREATE PROCEDURE statement for details, including examples.

Combining a result message with an input message when fetching data from external systems

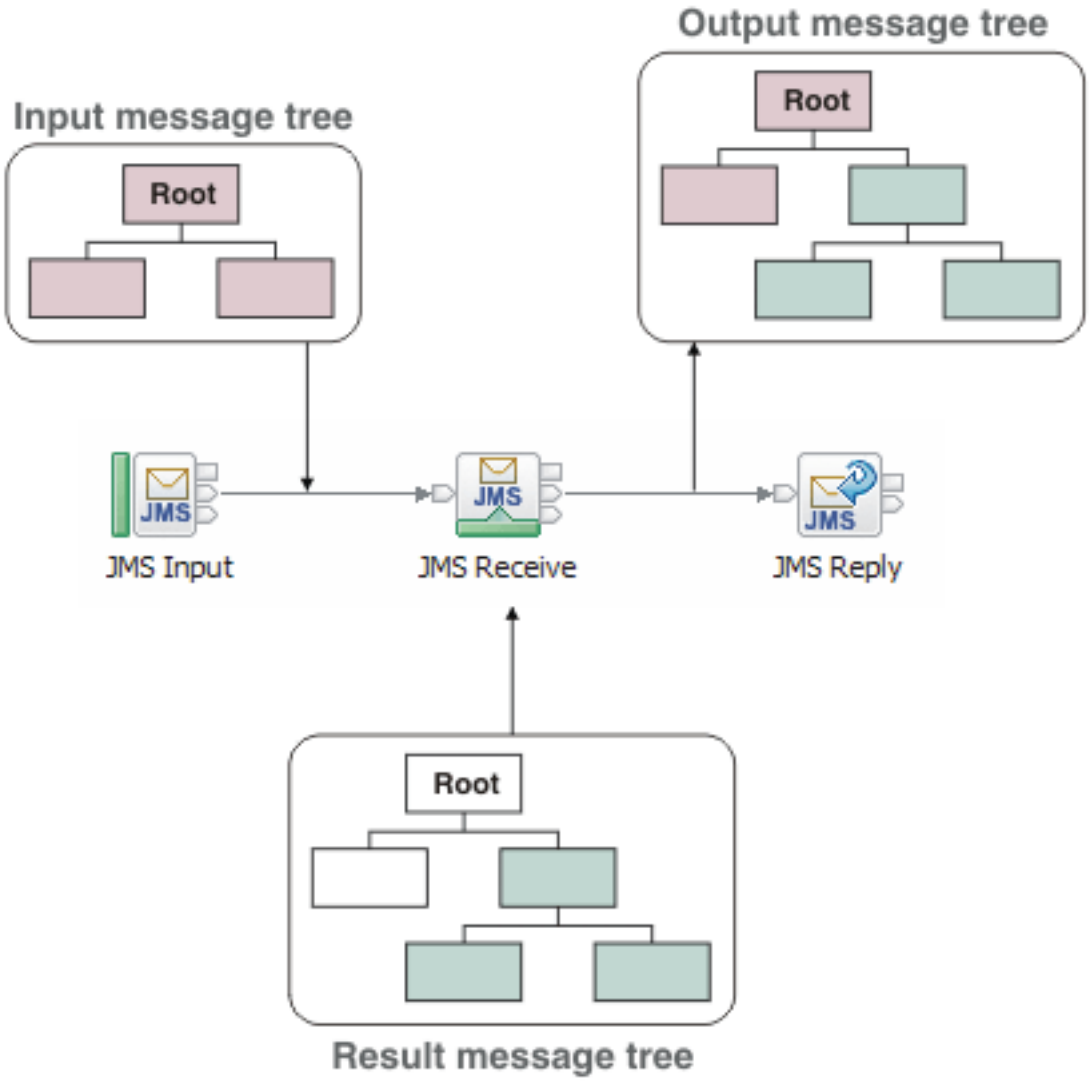
Some nodes can fetch data in the middle of a message flow, and merge the result data with the input message to augment or overwrite sections of the input message.

The following nodes can fetch data in the middle of a message flow and merge the result with the input message:

- Use the CICSRequest node to call CICS Transaction Server for z/OS programs over TCP/IP-based IP InterCommunications protocol (IPIC).

- Use the CORBARequest node to call an external CORBA application over Internet Inter-Orb Protocol (IIOP).
- Use the FileRead node to read one record, or the entire contents of a file, from within a message flow.
- Use the IMSRequest node to send a request to run a transaction on a local or remote IMS system, and wait for a response.
- Use the JMSReceive node to consume or browse JMS messages from a JMS queue in the middle of a message flow.
- Use the LoopBackRequest node to issue synchronous requests through a LoopBack connector, to create, retrieve, update, and delete data in a backend data source.
- Use the RESTAsyncRequest node with the RESTAsyncResponse node to construct a pair of message flows that interact asynchronously with an external REST API.
- Use the RESTRequest node to issue synchronous requests to external REST APIs.
- Use the SalesforceRequest node to send a request to a Salesforce system, to create, update, retrieve, and delete Salesforce data.
- Use the TCPIPClientReceive node to receive data over a client TCP/IP connection.
- Use the TCPIPServerReceive node to receive data over a server TCP/IP connection.

When you use one of these nodes in the middle of a message flow, you can fetch data from external resources. The result data is merged with the input message according to the values of the `Output data location` and, if it exists on the node, the `Result data location` properties.



The input root is first copied to the output root, and the result data is then copied to the location on the output tree specified by `Output data location`. The default value is `$OutputRoot`, which replaces the copied message tree with the result data, and propagates none of the input message.

The default value for `Result data location`, if it exists on the node, is `$ResultRoot`, which copies the entire result message to the output data location. If you specify a value underneath `$ResultRoot`, the specified subtree of the result message is inserted into the output tree.

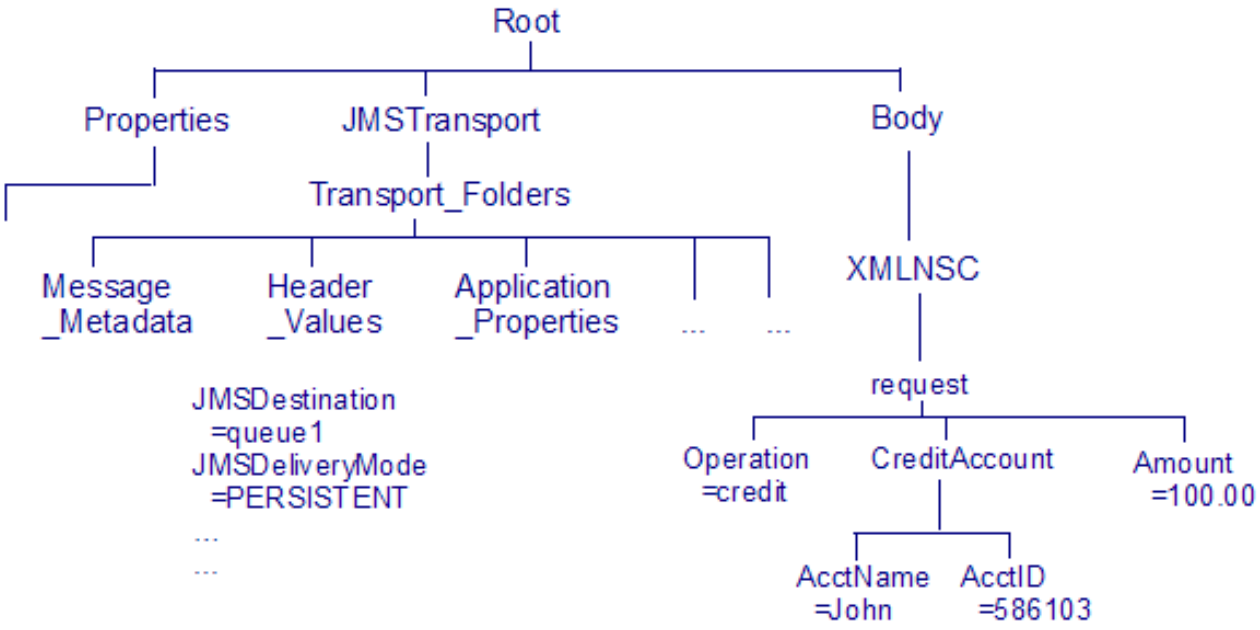
The following examples use the JMSReceive node and show the effect of possible different values specified for `Output data location` and `Result data location`.

Example 1: Replacing the whole message tree

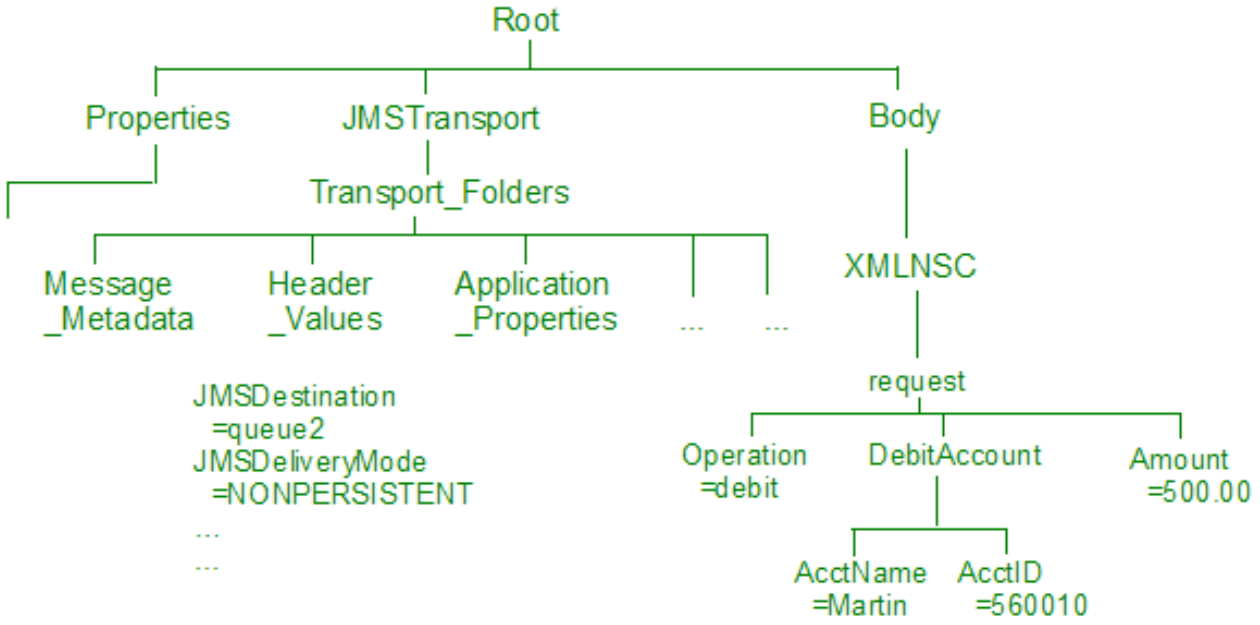
- `Output data location` = `$OutputRoot`
- `Result data location` = `$ResultRoot`

These values are the default values, and cause the input message to be overwritten with the result message.

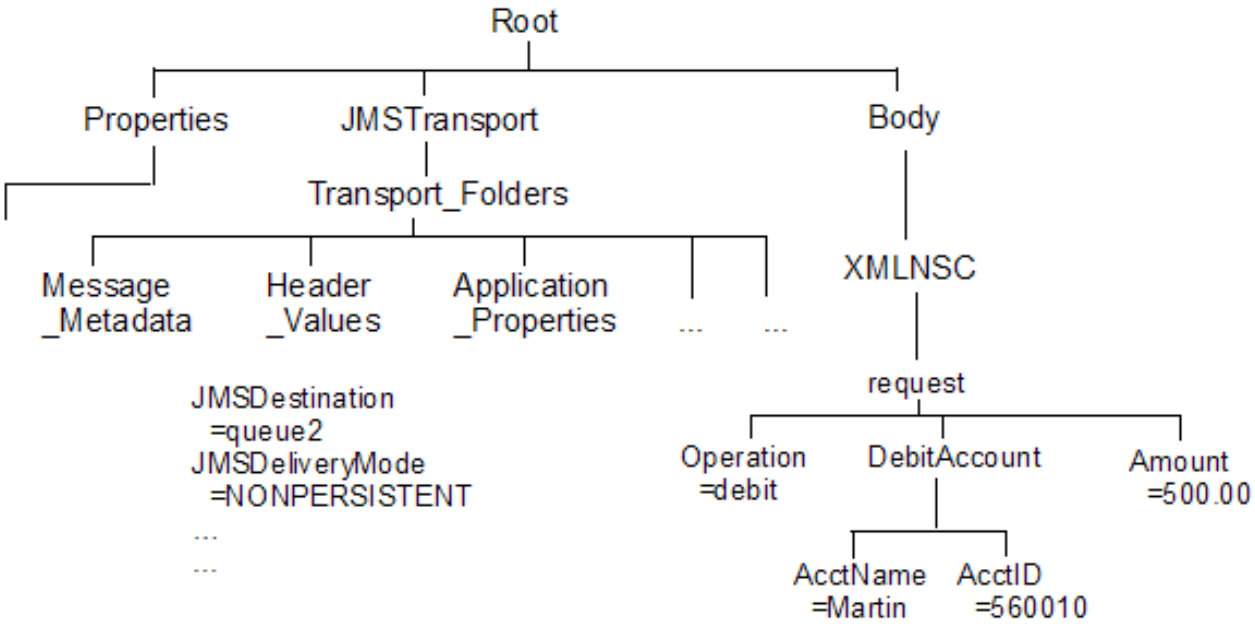
The input message tree is shown. The root element is selected to be overwritten:



The result message tree is shown. The root element is selected for insertion:



The output message tree is shown. The output root has been replaced by the result root tree:

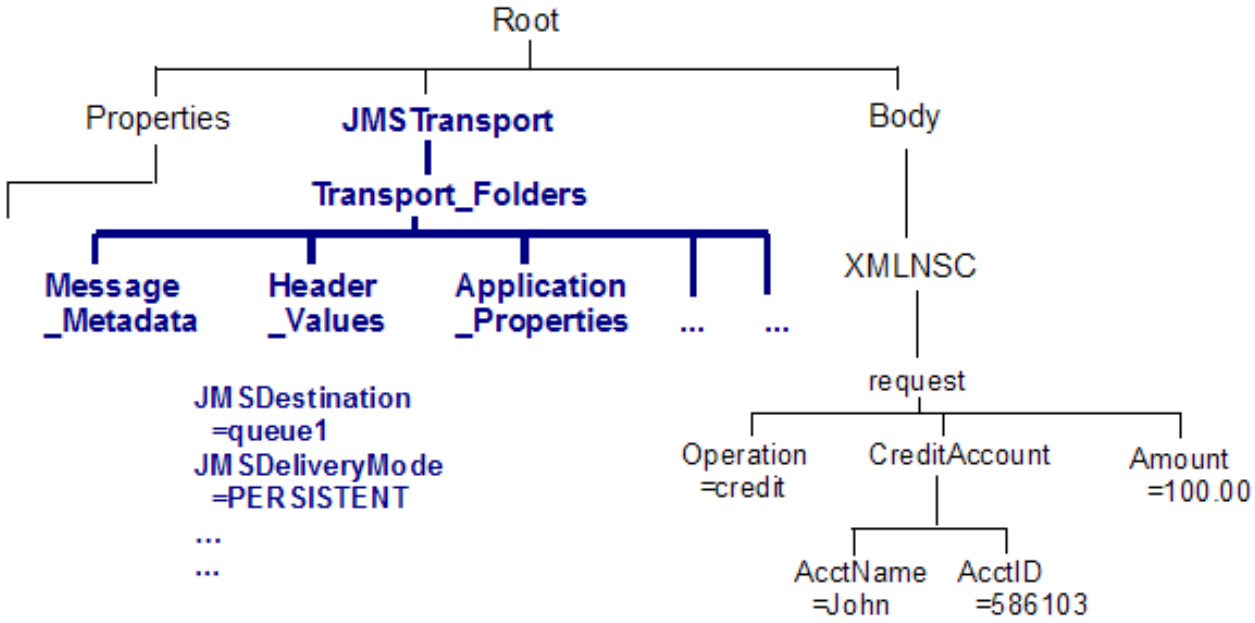


Example 2: Replacing the message headers

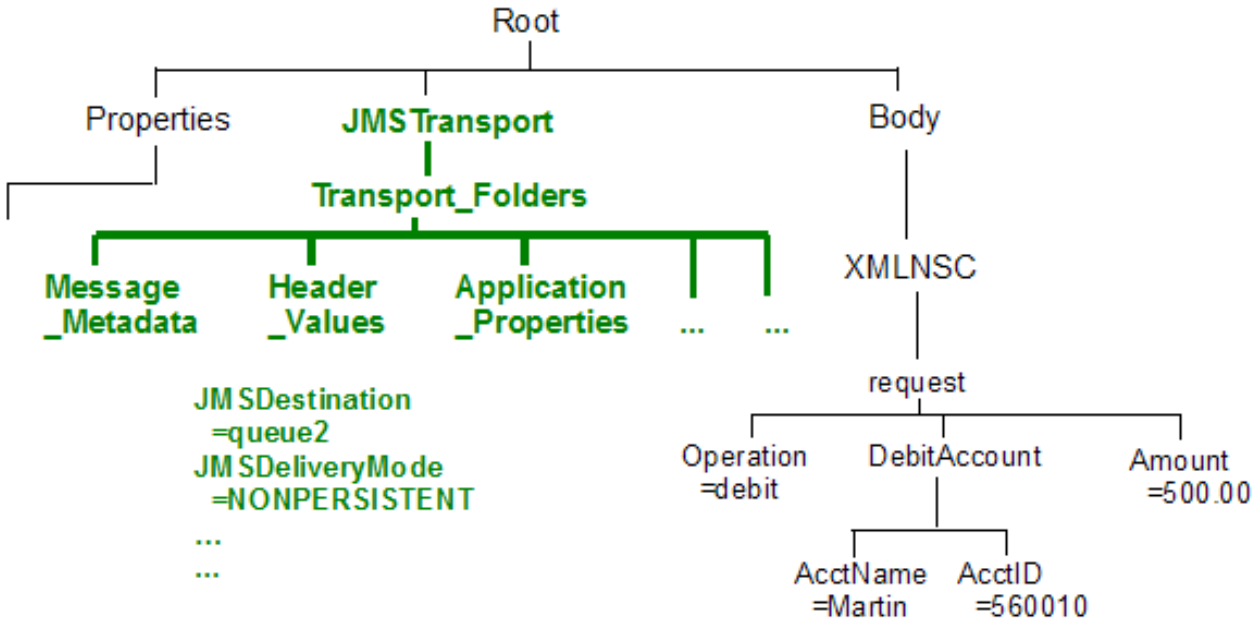
- `Output data location` = `$OutputRoot/JMSTransport`
- `Result data location` = `$ResultRoot/JMSTransport`

The input root is copied to the output root, and then the headers are overwritten by the headers of the result message.

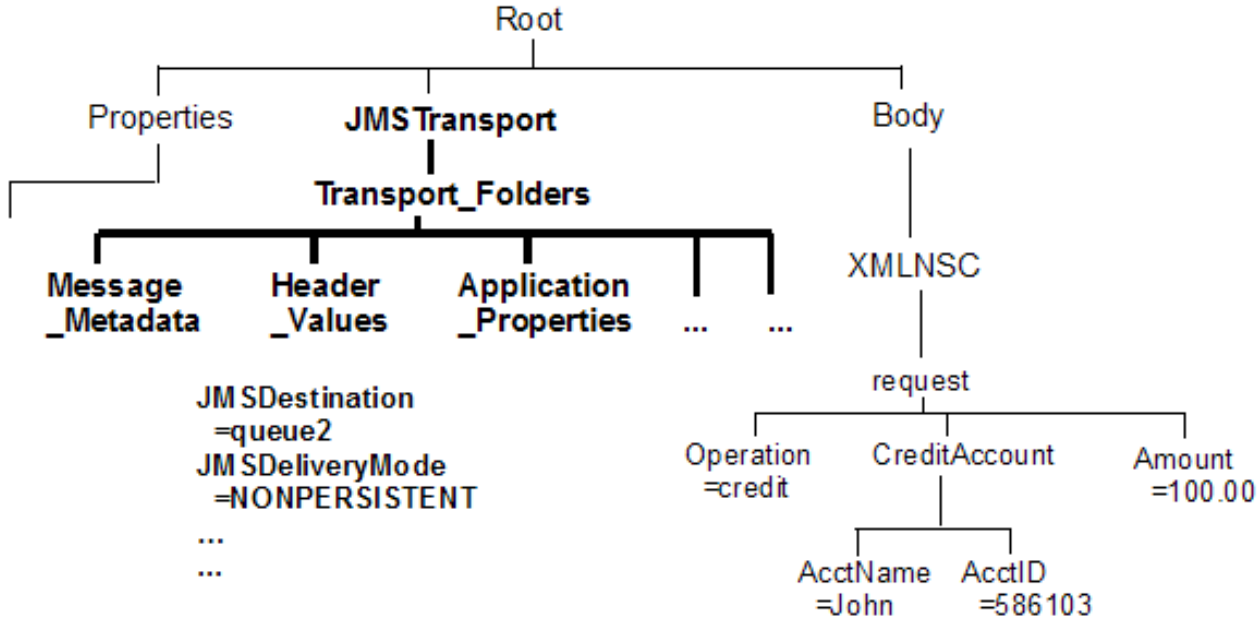
In the input message tree, the JMSTransport subtree is selected to be overwritten:



In the result message tree, the JMSTransport subtree is selected for insertion:



In the output message tree, the output JMSTransport subtree is replaced by the result JMSTransport subtree:

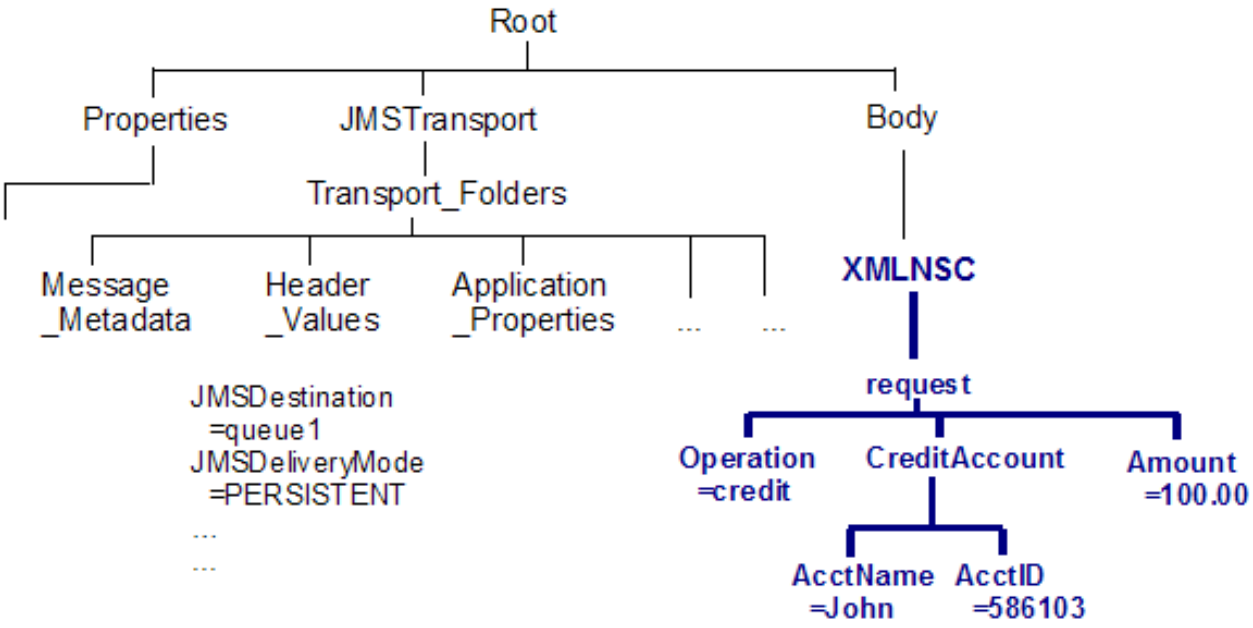


Example 3: Replacing the message body

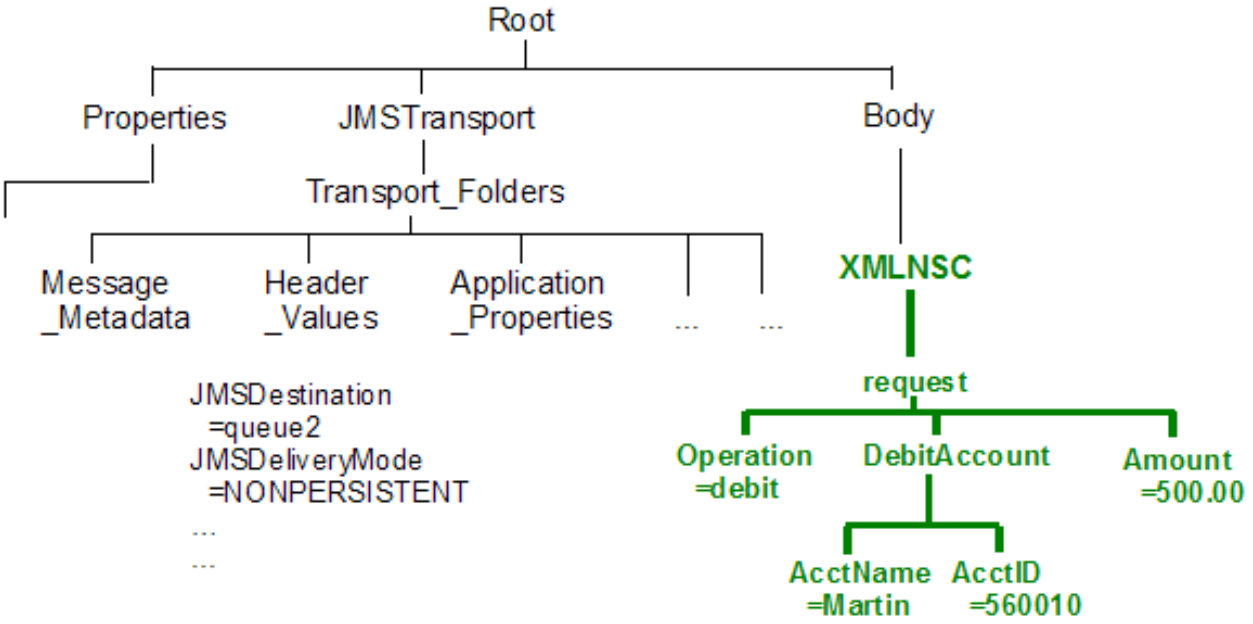
- `Output data location` = `$OutputRoot/XMLNSC`
- `Result data location` = `$ResultRoot/XMLNSC`

The input root is copied to the output root, and then the message body is overwritten by the body of the result message.

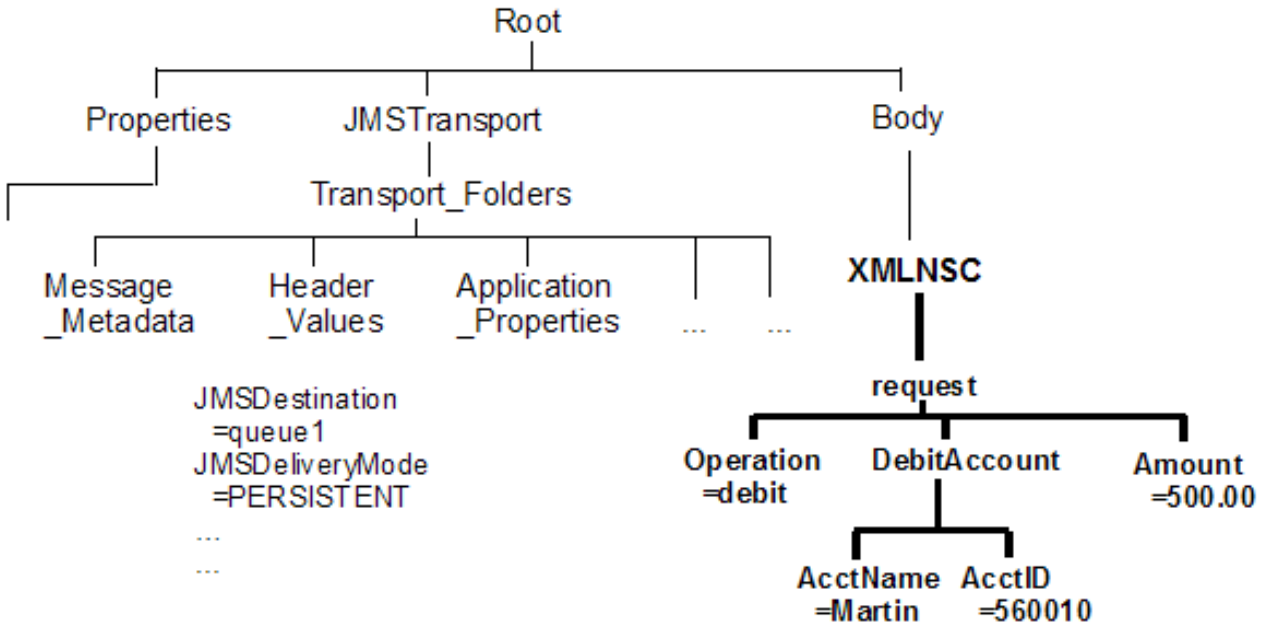
In the input message tree, the XMLNSC element is selected to be overwritten:



In the result message tree, the XMLNSC element is selected for insertion:



In the output message tree, the XMLNSC element has been replaced by the result XMLNSC element:

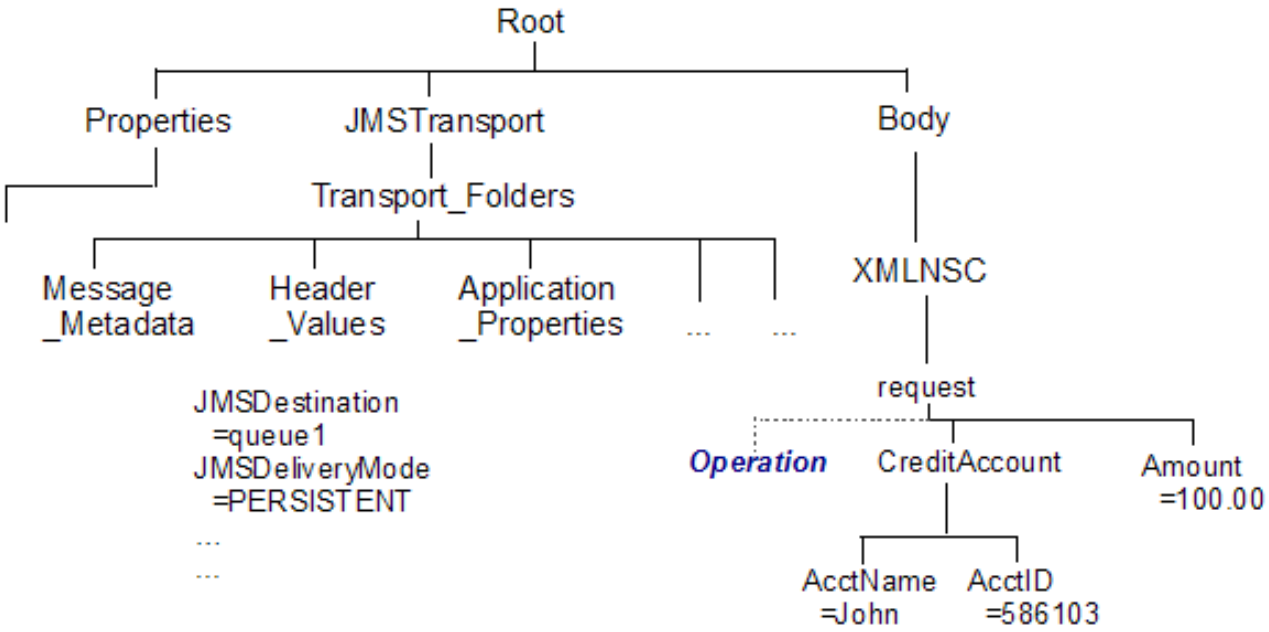


Example 4: Inserting a subtree of the result message

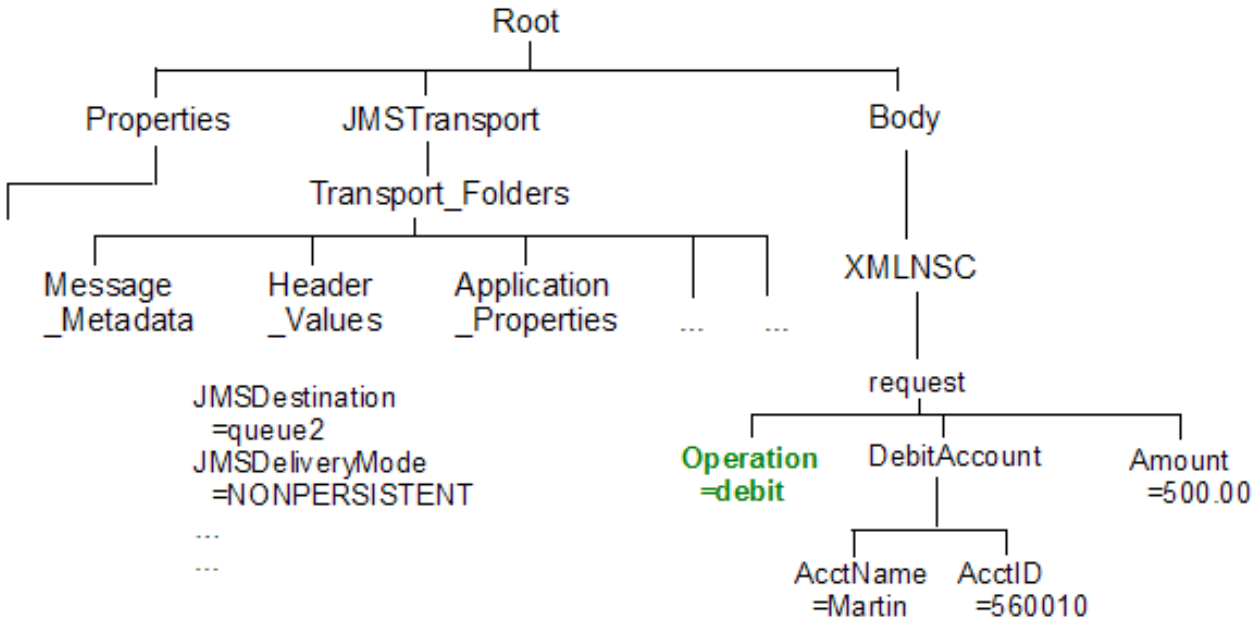
- `Output data location` = `$OutputRoot/XMLNSC/request/Operation`
- `Result data location` = `$ResultRoot/XMLNSC/request/Operation`

The input root is copied to the output root, and then the Operation subtree of the result message is inserted into the output tree underneath the request element. The input message tree does not contain an Operation element, but its location is determined by the `Output data location` value.

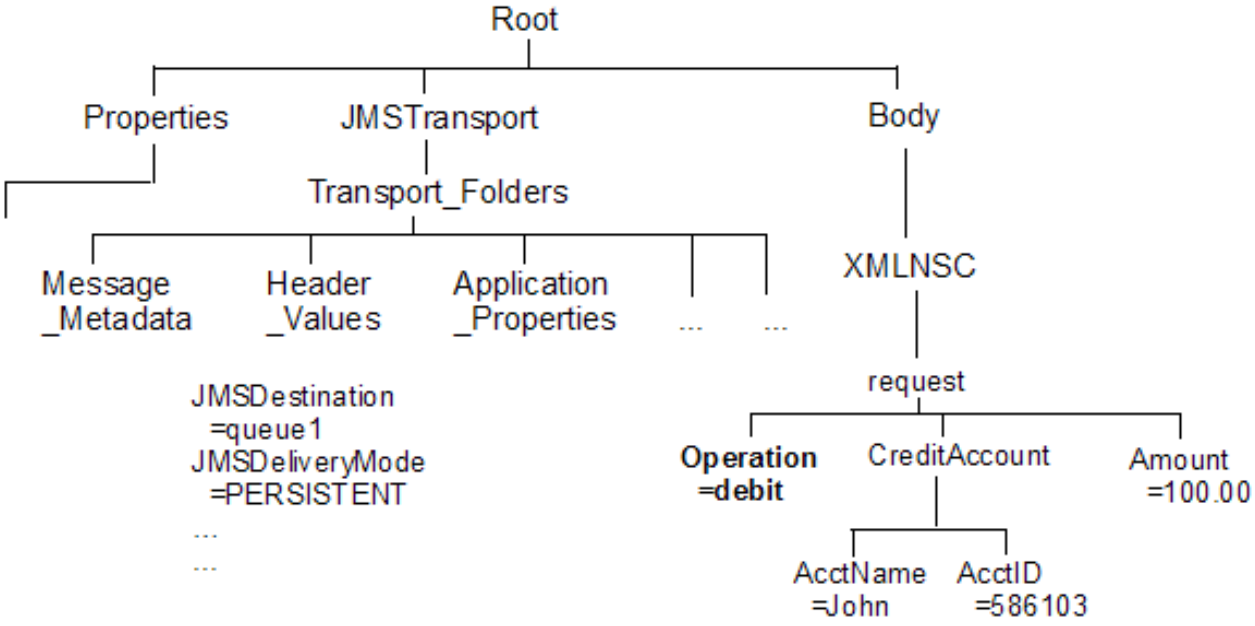
In the input message tree, the location is selected to be written to. This location does not yet exist:



In the result message tree, the Operation element is selected for insertion:



In the output message tree, the Operation element from the result message tree is inserted under the existing request element:



Processing events

You can use event driven message processing to control the flow of messages through your message flows, by using aggregation, message collections, message sequences, and timeout flows. Information about the state of in-flight messages is held on system queues that are controlled by IBM MQ queue managers.

About this task

If you want to use event driven processing on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node. The system queues that hold the state information are owned by the queue manager that is specified on the integration server. You associate a queue manager with the integration server by specifying the queue manager name on the

defaultQueueManager property in the `server.conf.yaml` configuration file; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

If you want to use event driven processing with an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as IBM App Connect Enterprise. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

You must create the system queues by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

The following topics contain information about using aggregation, message collections, message sequencing, and timeout flows:

- "Using aggregation" on page 1820
- "Using message collections" on page 1840
- Using message sequences
- "Configuring timeout flows" on page 1875

IBM MQ is not provided as part of the IBM App Connect Enterprise installation package, but your license for IBM App Connect Enterprise entitles you to install and use IBM MQ. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96 and "Enhanced flexibility in interactions with IBM MQ " on page 26. For more information about IBM MQ, see IBM MQ product information.

Using aggregation

Use aggregation to generate multiple requests from a single input message, and coordinate the multiple responses to provide a single consolidated response to that input message. Information about the state of in-flight messages is held on system queues that are controlled by IBM MQ.

About this task

If you want to use aggregation on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node. The system queues that hold the state information are owned by the queue manager that is specified on the integration server. You associate a queue manager with the integration server by specifying the queue manager name on the **defaultQueueManager** property in the `server.conf.yaml` configuration file; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

You must create the system queues by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

If you want to use aggregation with an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

The following topics describe the benefits of message flow aggregation, and explain how to configure your message flows to support message aggregation:

- "Message flow aggregation" on page 1821
- "Configuring aggregation flows" on page 1822

Message flow aggregation

Aggregation is the generation and fan-out of related requests that are derived from a single input message, and the fan-in of the corresponding replies to produce a single aggregated reply message.

The initial request that is received by the message flow, representing a collection of related request items, is split into the appropriate number of individual requests to satisfy the subtasks of the initial request. This process is known as *fan-out*, and it is provided by a message flow that includes aggregation nodes.

Replies from the subtasks are combined and merged into a single reply, which is returned to the original requester (or another target application) to indicate completion of the processing. This process is known as *fan-in*, and it is also provided by a message flow that includes aggregation nodes.

A message aggregation is initiated by a message flow that contains the AggregateControl node followed by an AggregateRequest node. The responses are aggregated back together using a flow that contains the AggregateReply node. The aggregation nodes work correctly only for transports that use a request/reply model; for example, IBM MQ Enterprise Transport.

IBM App Connect Enterprise provides three message flow nodes that support aggregation:

- The AggregateControl node
- The AggregateRequest node
- The AggregateReply node

When you include these nodes in your message flows, the multiple fan-out requests are issued in sequence from one thread in the integration server process.

If you use IBM MQ Enterprise Transport, the responses that are received by the fan-in flow must be valid reply messages that contain the reply identifier. You must set the reply identifier to the value of the message identifier field (MsgId) in the request message's message descriptor (MQMD), then store the reply identifier in the correlation identifier field (CorrelId) of the MQMD. If the CorrelId is set to MQCI_NONE, the AggregateReply node issues an error because the reply message is not valid, and cannot be matched to a request message.

You can also use these aggregation nodes to issue requests to applications outside the integration server environment. Messages can be sent asynchronously to external applications or services; the responses are retrieved from those applications, and the responses are combined to provide a single response to the original request message.

These nodes can help to improve response time because slow requests can be performed in parallel, and they do not need to follow each other sequentially. If the subtasks can be processed independently, and they do not need to be handled as part of a single unit of work, they can be processed by separate message flows.

You can design and configure a message flow that provides a similar function without using the aggregation nodes, by issuing the subtask requests to another application (for example, using the HTTPRequest node), and recording the results of each request in the local environment. After each subtask has completed, merge the results from the LocalEnvironment in a Compute node, and create the combined response message for propagating to the target application. However, all the subtasks are performed sequentially, and they do not provide the performance benefits of parallel operation that you can achieve if you use the aggregation nodes.

The aggregation nodes store state for aggregations on IBM MQ queues. By default, the following storage queues are used:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

By default, the timeout on the AggregateControl node is set to 0. If you do not specify a timeout (or if you leave it set to 0), aggregation requests that IBM MQ stores are never deleted unless all reply messages

are returned. This situation might lead to a gradual build up of messages on the internal queues. Set the timeout to a value greater than zero to ensure that requests are removed and that queues do not fill up with redundant requests. It is good practice to set the timeout value to a large value, for example, 86400 seconds (24 hours), so that the queues clear old aggregation messages even if timeouts are not required or expected.

You can set the timeout either by setting the `Timeout` property on the AggregateControl node, or by using an Aggregation policy and specifying the `Aggregation timeout` property. For more information, see "Setting timeout values for aggregation" on page 1831.

The aggregation nodes use IBM MQ message expiry to manage timeout of messages. For message expiry to work, the aggregation nodes must browse the message queues. The aggregation nodes browse the queues automatically to ensure that expired messages are processed.

Configuring aggregation flows

Use aggregation message flows to generate and fan-out a number of related requests, fan-in the corresponding replies, and compile those replies into a single aggregated reply message, by using the AggregateControl, AggregateRequest, and AggregateReply nodes.

Before you begin

- Read the concept topic on "Message flow aggregation" on page 1821.
- If you are using aggregation on an integration server that is managed by an integration node, ensure that you have installed IBM MQ. Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ, and you must install IBM MQ on the same computer as your integration node if you want to use the capabilities provided by the aggregation nodes. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

  If you are using aggregation on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

About this task

By default, the messages are put onto a set of default storage queues (beginning SYSTEM.BROKER.AGGR) for processing, but you can use an Aggregation policy to specify alternative queues. You can also use the Aggregation policy to set a timeout for the aggregation.

For an overview of using aggregation in message flows, see "Message flow aggregation" on page 1821.

To configure aggregation flows see the following topics:

- "Creating the aggregation fan-out flow" on page 1823
- "Creating the aggregation fan-in flow" on page 1826
- "Associating fan-out and fan-in aggregation flows" on page 1829
- "Setting timeout values for aggregation" on page 1831
- "Processing timed out aggregation messages" on page 1833
- "Using multiple AggregateControl nodes" on page 1833
- "Correlating input request and output response aggregation messages" on page 1834
- "Using control messages in aggregation flows" on page 1835
- "Handling exceptions in aggregation flows" on page 1837
- "Configuring the storage of events for aggregation nodes " on page 229

Creating the aggregation fan-out flow
The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

Before you begin

- For background information, see "Message flow aggregation" on page 1821.

About this task

You can include the fan-out and fan-in flow in the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 1829.

To create the fan-out flow:

Procedure

1. Create a message flow to provide the fan-out processing.
   For more information, see "Creating a message flow" on page 574.
2. Add the following nodes to the message flow, then configure and connect them as described.

   Input node
   The input node receives an input message from which multiple request messages are generated. This node can be any one of the built-in nodes, or a user-defined input node.

   a. Select the input node to open the Properties view. The node properties are displayed.
   b. Specify the source of input messages for this node. For example, specify the name of an IBM MQ queue in the `Queue name` property. The MQInput node retrieves messages from this queue.
   c. Optional: set values for any other properties that you want to configure for this node. For example, to ensure that aggregate request messages are put under sync point, set the `Transaction mode` property to `Yes`. This option avoids the situation where the AggregateReply node receives response messages before it has received the control message that informs it of the aggregation instance. Putting the fan-out flow under transactional control ensures that the fan-out flow completes before any response messages get to the AggregateReply.
   d. Connect the input node's Out terminal to the In terminal of an AggregateControl node. This option represents the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateControl node. For example, store the request for audit purposes (in a Database node), or add a unique identifier to the message (in a Compute node).
   e. Optional: if your fan-out and fan-in flows are combined in one message flow, modify the `Order mode` property on the `Advanced` tab. Select the `By Queue Order` option and ensure that the `Logical Order` property is also selected. These options force the input node to be single threaded to maintain the logical order of the messages that arrive on the queue. Any additional instance threads that you make available are then shared among only the fan-in input nodes to improve the performance of aggregation. If your fan-in and fan-out flows are in separate message flows this step is not required because you can make additional threads available specifically to the fan-in flow.

   AggregateControl node
   The AggregateControl node updates the local environment that is associated with the input message with information required by the AggregateRequest node. The AggregateControl node creates the `LocalEnvironment.ComIbmAggregateControlNode` folder. This folder and the

fields in it are for internal use by IBM App Connect Enterprise and you should not rely on their existence or values when developing your message flows.

a. Select the AggregateControl node to open the Properties view. The node properties are displayed.

b. Set the `Aggregate name` property of the AggregateControl node to identify this particular aggregation. It is used later to associate this AggregateControl node with a specific AggregateReply node. The `Aggregate name` that you specify must be contextually unique in an integration node.

c. Optional: set the `Timeout` property to specify how long the integration node waits for replies to arrive before taking action (described in "Setting timeout values for aggregation" on page 1831). If a timeout is not set on the AggregateControl node then aggregate requests stored internally will not be removed unless all aggregate reply messages return. This situation might lead to a gradual build-up of messages on the internal queues. To avoid this situation, set the timeout to a value other than zero (zero means that a timeout never occurs) so that when the timeout is reached the requests are removed and the queues do not fill up with redundant requests. Even if timeouts are not required or expected, it is good practice to set the timeout value to a large value; for example, 86400 seconds (24 hours) so that the queues occasionally get cleared of old aggregations.

d. Connect the Out terminal of the AggregateControl node to the In terminal of one or more Compute nodes that provide the analysis and breakdown of the request in the input message that is propagated on this terminal.

Compute node

The Compute node extracts information from the input message and constructs a new output message.

If the target applications that handle the subtask requests can extract the information that they require from the single input message, you do not need to include a Compute node to split the message. You can pass the whole input message to all target applications.

If your target applications expect to receive an individual request, not the whole input message, you must include a Compute node to generate each individual subtask output message from the input message. Configure each Compute node in the following way, copying the appropriate subset of the input message to each output message:

a. Select the Compute node to open the Properties view. The node properties are displayed.

b. Select a value for the Basic property `Compute mode`. This property specifies which sections of the message tree are modified by the node. The AggregateControl node inserts elements into the local environment tree in the input message that the AggregateRequest node reads when the message reaches it. Ensure that the local environment is copied from the input message to the output message in the Compute node. This configuration happens automatically

unless you specify a value that includes local environment (one of All, LocalEnvironment, LocalEnvironment and Message, or Exception and LocalEnvironment).

If you specify one of these values, the integration node assumes that you are customizing the Compute node with ESQL that writes to local environment, and that you are copying any elements in that tree that are required in the output message.

To modify the local environment, add the following statement to copy the required aggregate information from input message to output message:

```
SET OutputLocalEnvironment.ComIbmAggregateControlNode =
            InputLocalEnvironment.ComIbmAggregateControlNode;
```

c. Optional: set values for any other properties that you want to configure for this node.

d. Connect the Out terminal of each Compute node to the In terminal of the output node that represents the destination of the output request message that you have created from the input message in this node.

Output node

Include an output node for each output message that you generate in your fan-out flow. Configure each node, as described later in this section, with the appropriate modifications for each destination.

The output node must be an output node that supports the request/reply model, such as an MQOutput node, or a mixture of these nodes (depending on the requirements of the target applications).

a. Select the output node to open the Properties view. The node properties are displayed.

b. Specify the destination for the output messages for this node. For example, specify the name of an IBM MQ queue in the `Queue name` property to which the MQOutput node sends messages. The target application must process its request, and send the response to the reply destination indicated in its input message (for example the IBM MQ ReplyToQueue).

c. Click **Request** in the left view and set values for these properties to specify that replies are sent to the input queue of the fan-in flow.

d. Optional: set values for any other properties that you want to configure for this node.

e. Connect the Out terminal of the output node to the In terminal of an AggregateRequest node. When the message is propagated through the output node's Out terminal, the built-in output node updates the WrittenDestination folder in the associated local environment with additional information required by the AggregateRequest node.

The information written by the built-in nodes is queue name, queue manager name, message ID, and correlation ID (from the MQMD), and message reply identifier (set to the same value as message ID).

AggregateRequest node

Include an AggregateRequest node for each output message that you generate in your fan-out flow.

a. Select the AggregateRequest node to open the Properties view. The node properties are displayed.

b. Set the Basic property `Folder name` to a value that identifies the type of request that has been sent out. This value is used by the AggregateReply node to match up with the reply message when it is received in the fan-in flow. The folder name that you specify for each request that the fan-out flow generates must be unique.

The AggregateRequest node writes a record in IBM MQ for each message that it processes. This record enables the AggregateReply node to identify which request each response is associated with. If your output nodes are non-transactional, the response message might arrive at the fan-in

flow before this IBM MQ update is committed. For details on how you can use timeouts to avoid this situation, see "Setting timeout values for aggregation" on page 1831.

CAUTION: Although the use of timeouts can help to avoid this situation described previously, configure your fan-out flow to be transactional. Therefore, response messages cannot get to the fan-in flow before the corresponding AggregateRequest nodes have committed their IBM MQ updates.

3. To save the message flow and validate its configuration, press **Ctrl-S** or click **File** > **Save**.

What to do next

To collect the aggregation responses initiated by your fan-out flow, create a fan-in flow, as described in "Creating the aggregation fan-in flow" on page 1826.

Creating the aggregation fan-in flow

The aggregation fan-in flow receives the responses to the request messages that are sent out by the fan-out flow, and constructs a combined response message containing all the responses received.

Before you begin

- Read an overview of aggregation in "Message flow aggregation" on page 1821.

About this task

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. For more information about the benefits of configuring separate message flows, see "Associating fan-out and fan-in aggregation flows" on page 1829. Do not deploy multiple copies of the same fan-in flow either to the same or to different integration servers.

If you do not configure the fan-out flow to be transactional, the timeout values that you have specified might result in the combined response message being generated before the fan-in flow has received all the replies. For more information, see "Creating the aggregation fan-out flow" on page 1823.

To create the fan-in flow:

Procedure

1. Create a message flow to provide the fan-in processing.
2. Add the following nodes in the editor view and configure and connect them as described:

Input node

The input node receives the responses to the multiple request messages that are generated from the fan-out flow.

This node must be an input node that supports the request/reply model, such as an MQInput node, or a mixture of these nodes (depending on the requirements of the applications that send these responses). The response that is received by each input node must be sent across the same protocol as the request to which it corresponds. For example, if you include an MQOutput node in

the fan-out flow, the response to that request must be received by an MQInput node in this fan-in flow.

a. Select the input node to open the Properties view. The node properties are displayed.

b. Specify the source of input messages for this node; for example, specify the name of an IBM MQ queue in the Basic property Queue name from which the MQInput node retrieves messages.

c. Optional: specify values for any other properties that you want to configure for this node.

d. Connect the Out terminal of the input node to the In terminal of an AggregateReply node.

   Connect the terminals in this way to create the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateReply node; for example, you might want to store the replies for audit purposes (in a Database node).

Include just one input node that receives all the aggregation response messages at the beginnings of the fan-in flow as described previously. If you include multiple input nodes, threads that are started by a specific reply input node might complete the aggregation and execution of the message flow while other threads are sending their response messages to the AggregateReply node and becoming eligible to timeout. Use a single input node to enable sequential processing of replies for each aggregation. Specify additional instances to provide greater processing throughput in this single node, see Configurable properties in a file.

AggregateReply node

The AggregateReply node receives the inbound responses from the input node through its In terminal. The AggregateReply node stores each reply message for subsequent processing.

When all the replies for a particular group of aggregation requests have been collected, the AggregateReply node creates an aggregated reply message, and propagates it through the Out terminal.

a. Select the AggregateReply node to open the Properties view. The node properties are displayed.

b. Set the Aggregate name property of the AggregateReply node to identify this aggregation. Set this value to be the same value that you set for the Aggregate name property in the corresponding AggregateControl node in the fan-out flow.

c. Optional: to retain an unrecognized message before propagating it to the Unknown terminal, set a value for the Unknown message timeout property. If you are using separate fan-out and fan-in flows, set this value to a non-zero number if the control message might be delayed.

d. Optional: to explicitly handle unrecognized messages, connect the Unknown terminal to another node, or sequence of nodes. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

e. Optional: if you have specified a timeout value for this aggregation in the AggregateControl node, and you want to explicitly handle timeouts that expire before all replies are received, connect the Timeout terminal to another node, or sequence of nodes. Partially complete aggregated replies are sent to the Timeout terminal if the timer expires. If you do not connect this terminal to another node in the message flow, messages propagated through this terminal are discarded.

f. Optional: specify values for any other properties that you want to configure for this node.

g. Connect the Out terminal of the AggregateReply node to the In terminal of a Compute node.

Attention: The Control terminal of the AggregateReply node was deprecated at Version 6.0, and by default any connections to this terminal (either direct or indirect) are ignored. This change maximizes the efficiency of aggregation flows and does not damage the reliability of aggregations. This configuration provides the optimum content.

However, if you want the AggregateReply node to receive, on its Control terminal, the control message that was sent by the corresponding AggregateControl node on the fan-out flow, you must make the necessary connections as described in “Creating the aggregation fan-out flow” on page 1823. Keep the path from the AggregateReply node to the output

node as direct as possible to maximize the performance of aggregations. Do not modify the content of this control message.

In addition, for the Control terminal and connections to it to be recognized, you must enable the environment variable MQSI_AGGR_COMPAT_MODE. If you choose this option, the performance and behavior of message aggregations might be affected; for a full description of these implications and the environment variable, see "Using control messages in aggregation flows" on page 1835.

Aggregated messages which are sent from the AggregateReply node output terminals (Out and Timeout) are not validated. Validation of data must be done before messages are sent to the AggregateReply node, because it ignores validation options when reconstructing the stored messages.

Compute node

The Compute node receives the message that contains the combined responses. Typically, the format of this combined message is not valid for output, because the aggregated reply message has an unusual structure and cannot be parsed into the bit stream required by some nodes, for example the MQOutput node. The Out and Timeout terminals always propagate an aggregated reply message, which always requires further processing before it can be propagated to an MQOutput node. Therefore you must include a Compute node and configure this node to create a valid output message.

a. Select the Compute node to open the Properties view. The node properties are displayed.
b. Specify in the Basic property ESQL module the name of the ESQL module that customizes the function of this node.
c. Right-click the node and click **Open ESQL** to open the ESQL file that contains the module for this node. The module is highlighted in the ESQL editor view.
d. Code the ESQL to create a single output message from the aggregated replies in the input message.

   The aggregated reply message is propagated through the Out terminal. Information about how you can access its contents is provided in "Accessing the combined message contents" on page 1828.
e. Optional: specify values for any other properties that you want to configure for this node.
f. Connect the Out terminal of the Compute node to the In terminal of the output node that represents the destination of the single response message.

Output node

Include an output node for your fan-in flow. This node can be any of the built-in nodes, or a user-defined output node.

a. Select the output node to open the Properties view. The node properties are displayed.
b. Specify the destination for the output message for this node; for example, specify in the Basic property Queue name the name of an IBM MQ queue to which the MQOutput node sends messages.
c. Optional: specify values for any other properties that you want to configure for this node.

3. To save the message flow and validate its configuration, press Ctrl-S or click **File** > **Save**.

Accessing the combined message contents

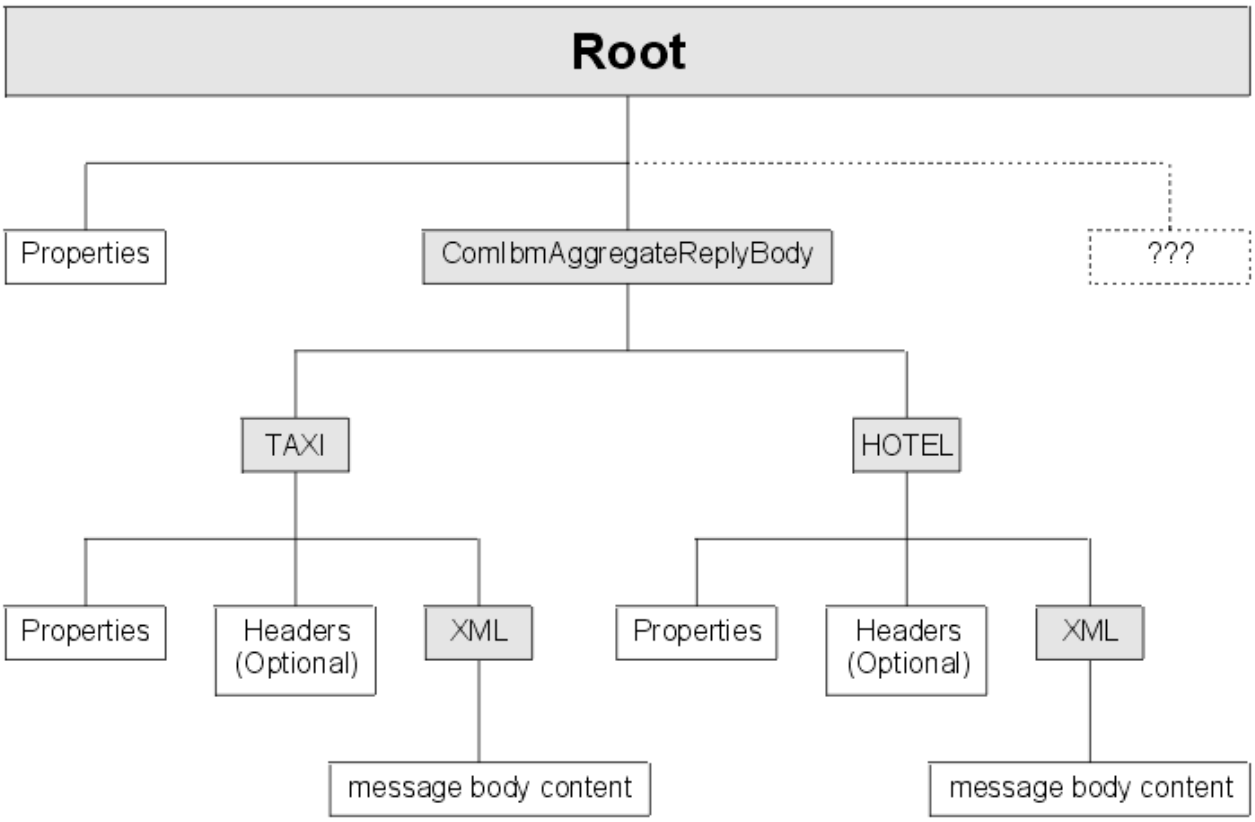
About this task

The AggregateReply node creates a folder in the combined message tree below Root, called ComIbmAggregateReplyBody. Below this folder, the node creates a number of subfolders using the names that you set in the AggregateRequest nodes. These subfolders are populated with the associated reply messages.

For example, the request messages might have folder names:

- TAXI
- HOTEL

The resulting aggregated reply message created by the AggregateReply node might have a structure like the following example:



Use ESQL within a Compute node to access the reply from the taxi company using the following correlation name:

```
InputRoot.ComIbmAggregateReplyBody.TAXI.xyz
```

The folder name does not have to be unique. If you have multiple requests with the folder name TAXI, you can access the separate replies using the array subscript notation, for example:

```
InputRoot.ComIbmAggregateReplyBody.TAXI[1].xyz
InputRoot.ComIbmAggregateReplyBody.TAXI[2].xyz
```

Associating fan-out and fan-in aggregation flows
Associate the fan-out message flow processing with its corresponding fan-in message flow processing by setting the `Aggregate Name` property of the AggregateControl and AggregateReply nodes in your aggregation flow to the same value.

Before you begin

If you did not configure this property when you created your fan-in and fan-out flows, you must complete this task.

You must have completed the following tasks:

- “Creating the aggregation fan-out flow” on page 1823
- “Creating the aggregation fan-in flow” on page 1826

About this task

The `Aggregate Name` must be contextually unique within an integration node. You can have only one AggregateControl node and one AggregateReply node with a particular `Aggregate Name`, although you can have more than one AggregateControl node with the same `Aggregate Name`, see "Using multiple AggregateControl nodes" on page 1833. Do not deploy a fan-in flow to multiple integration servers on the same integration node; results are unpredictable.

You can either create the fan-out and fan-in flows in the same message flow, or in two different message flows. In either case, the two parts of the aggregation are associated when you set the `Aggregate Name` property.

How you configure your aggregation flow depends on a number of factors:

- The design of your message flow.
- The hardware on which the integration node is running.
- The timeout values that you choose, see "Setting timeout values for aggregation" on page 1831.
- How you expect to maintain the message flows.

You can include the fan-out and fan-in flow within the same message flow. However, you might prefer to create two separate flows. The advantages of creating separate fan-out and fan-in flows are:

- You can modify the two flows independently.
- You can start and stop the two flows independently.
- You can deploy the two flows to separate integration servers to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.
- You can allocate different numbers of additional threads to the two flows, as appropriate, to maintain a suitable processing ratio.

To associate the fan-out flow with the fan-in flow:

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow that contains your fan-out flow.
3. Select the AggregateControl node to open the Properties view.

   The node properties are displayed.
4. Set the `Aggregate Name` property of the AggregateControl node to identify this aggregation. The `Aggregate Name` that you specify must be contextually unique within an integration node.
5. `If you have separate fan-out and fan-in flows:`
   a) Press Ctrl-S or click **File** > **Save *name*** on the taskbar menu (where *name* is the name of this message flow) to save the message flow and validate its configuration.
   b) Open the message flow that contains your fan-in flow.
6. Select the AggregateControl node to open the Properties view.

   The node properties are displayed.
7. Set the `Aggregate Name` property of the AggregateReply node to the same value that you set for the `Aggregate Name` property in the corresponding AggregateControl node in the fan-out flow.
8. Press Ctrl-S or click **File** > **Save *name*** to save the message flow and validate its configuration.

What to do next

In IBM App Connect Enterprise, fan-out and fan-in flows were also associated by sending control messages from the AggregateControl node to the AggregateReply node. This facility is no longer available. For optimum performance, do not connect the AggregateControl and AggregateReply node. However, if you do want to use control messages in your aggregations, and you want to connect these two nodes, see "Using control messages in aggregation flows" on page 1835.

Setting timeout values for aggregation
You can use two properties of the aggregation nodes to set timeout values for aggregated message processing.

Before you begin

To complete this task, you must have completed the following tasks:

- "Creating the aggregation fan-out flow" on page 1823
- "Creating the aggregation fan-in flow" on page 1826

About this task

Two situations might require the use of timeouts:

- The need to receive an aggregated reply message within a specified time. For this situation, you set the `Timeout` property of the AggregateControl node.
- The need to wait before propagating an unrecognized message to the Unknown terminal. For this situation, you set the `Unknown Message Timeout` property on the AggregateReply node.

The following sections describe each situation in more detail.

To receive an aggregated reply message within a specified time

About this task

In some situations, you might need to receive an aggregated reply message within a specified time. Some reply messages might be slow to return, or might never arrive. For these situations, open the fan-out message flow and set the timeout interval by using one of the following steps:

Directly on the AggregateControl node

Set the `Timeout` property of the AggregateControl node to specify how long (in seconds) the integration node must wait for replies. By default, this property is set to 0 (zero), which means that there is no timeout and the integration node waits indefinitely. For more information, see node.

Using an Aggregation policy

You can also use an Aggregation policy to specify the timeout interval. To use an Aggregation policy, set the `Aggregate name` property of the AggregateControl node to match the policy project and policy name (in the format *"{PolicyProjectName}:PolicyName"*), and specify the timeout value in the `Aggregation timeout` property of the policy.

To specify a default Aggregation policy for all message flows that are deployed to an integration server, set the **Aggregation** property in the `server.conf.yaml` file to the name of an Aggregation policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*

The value of the `Aggregation timeout` property of the policy overrides the `Timeout` property of the AggregateControl node.

Unlike the `Timeout` property of the AggregateControl node, you can set the `timeoutSeconds` property value to one decimal place, allowing timeout intervals of less than one second if required. Values of 0.5, 1.7, and 100.1 are all valid, but a value of 0.22 is invalid.

When setting timeout intervals of less than one second, it is recommended that the `queuePrefix` property is set up and the `timeoutThreads` property is set to a minimum of 2. This is to make sure that the timeout processing has sufficient resources to meet the timeout requirements. Despite this, users may well encounter other unexpected operational factors that could impact on the ability to run timeout intervals below the one-second requirement.

For more information about the Aggregation policy, see Aggregation policy.

Dynamically from an incoming message

You can use the `Timeout location` property of the AggregateControl node to specify the location of a timeout value in the incoming message. Any timeout value that is specified in this way overrides the values that are specified by the AggregateControl node and the Aggregation policy.

In the same way, as for the Aggregation policy, the timeout value is specified down to one decimal place.

For more information, see node.

The timeout precedence is determined by values being set in the message, the policy, or the node, in the following order:

1. If a timeout value is specified in the incoming message (in the location that is specified by the `Timeout location` property of the AggregateControl node) this value is used.
2. If no location is specified by the `Timeout location` property, or if the location in the message is empty or does not exist, the value that is specified by the `timeoutSeconds` property of the Aggregation policy is used (if one exists).
3. If no Aggregation policy has been defined, or if the `Aggregation timeout` property of the policy is not set, the value that is set in the `Timeout` property of the AggregateControl node is used.

By default, timeout polling occurs every 5 seconds. Therefore, if you set the `Timeout` property to a value that is not a multiple of 5, an extra delay occurs. For example, if you set the `Timeout` property to 7 seconds, you see a delay of 3 seconds until timeout polling next occurs. You can change the default timeout polling interval by using the environment variable MQSI_AGGR_WAIT_TIMEOUT. Valid values are 1000 - 5000 milliseconds. To change the default polling interval, stop the integration node, then restart the integration node in an environment where you have set the MQSI_AGGR_WAIT_TIMEOUT environment variable.

If a `Timeout` property value has been set to less than one second the timeout polling interval rules are ignored. In this instance the timeout polling interval is automatically calculated as part of timeout processing. Therefore, no manual intervention is required in order to update the environment variable MQSI_AGGR_WAIT_TIMEOUT.

If the timeout interval passes without all the replies arriving, the replies that have arrived are turned into an aggregated reply message by the corresponding AggregateReply node, and propagated to its Timeout terminal. You can process this partial response message in the same way as a complete aggregated reply message. If you prefer, you can provide special processing for incomplete aggregated replies.

To wait before propagating an unrecognized message to the Unknown terminal

About this task

When a message arrives at the In terminal of an AggregateReply node, it is examined to see if it is an expected reply message. If it is not recognized, the message is propagated to the Unknown terminal. You might want the integration node to wait for a specified time period before propagating the message for the following reasons:

- The reply message might arrive before the work completed by the AggregateRequest node has been transactionally committed. This situation can be avoided by configuring the `Transaction mode` property of the input node, as described in “Creating the aggregation fan-out flow” on page 1823.
- The reply message might arrive before the control message. This situation can be avoided by leaving the Control terminal of the AggregateControl node unconnected. For more information about the implications of connecting the Control terminal, see “Using control messages in aggregation flows” on page 1835.

These situations are most likely to happen if you send the request messages out of syncpoint, and might result in valid replies being sent to the Unknown terminal. To reduce the likelihood of this event, complete the following steps.

1. Open the fan-in message flow.
2. Set the `Unknown Message Timeout` property of the AggregateReply node.

   When you set this property, a message that cannot be recognized immediately as a valid reply is held persistently in the integration node for the number of seconds that you specify for this property.

If the unknown timeout interval expires, and the message is recognized, it is processed. The node also checks to see if this previously unknown message is the last reply that is needed to make an aggregation complete. If it is, the aggregated reply message is constructed and propagated.

If the unknown timeout interval expires and the message is still not recognized, the message is propagated to the Unknown terminal.

Processing timed out aggregation messages
Assign additional processing threads to enable processing of timed out aggregation messages in the AggregateReply node.

Before you begin

To complete this task, the following conditions must be met:

- The AggregateReply node must be using an Aggregation policy.
- The associated Aggregation policy must have a queue prefix set that is unique across all Aggregation policies.
- The set of aggregation queues referred to by the associated Aggregation policy must only be used by a single integration server.

About this task

By default the AggregateReply node uses a single thread to process timed out aggregation messages. In scenarios where a high volume of messages are expected to timeout, a backlog of messages can accumulate on the SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT queue. Follow this task to assign additional processing threads for timeout processing in the AggregateReply node.

You can configure the number of threads that are used to process timeout messages by setting the `Number of timeout threads` property of the associated Aggregation policy.

Using multiple AggregateControl nodes
You might find it useful to design a fan-out flow with multiple AggregateControl nodes, all with the same value set for the property `Aggregate Name`, but with different values for the `Timeout` property. You can reuse an aggregate name in only this situation.

Before you begin

For background information, see “Message flow aggregation” on page 1821.

About this task

You might want to use multiple AggregateControl nodes if, for example, you have created an aggregation flow that books a business trip, and you might have some requests that need a reply within two days. However, other, more urgent requests, need a reply within two hours.

To configure an aggregation flow that uses multiple AggregateControl nodes, complete the following steps.

Procedure

1. Create or open the fan-out message flow.

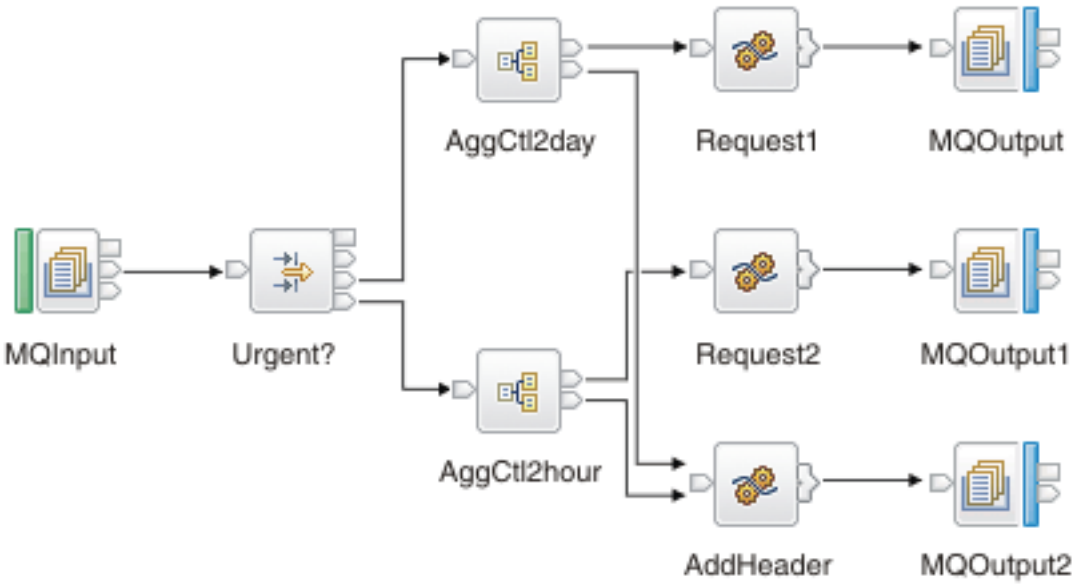
2. Configure the required number of AggregateControl nodes. Set the Basic property `Aggregate Name` of each node to the same value. For example, include two nodes and enter the name `JOURNEY` as the `Aggregate Name` for both.
3. Set the value for the `Timeout` property in each node to a different value. For example, set the `Timeout` in one node to two hours; set the `Timeout` in the second node to two days.
4. Configure a Filter node to receive incoming requests, check their content, and route them to the correct AggregateControl node.
5. Connect the nodes together to achieve the required result.

   For example, if you have configured the Filter node to test for requests with a priority field set to urgent, connect the true terminal to the AggregateControl node with the short timeout. Connect the false terminal to the AggregateControl node with the longer timeout. Connect the out terminals of the AggregateControl nodes to the following nodes in the fan-out flow.

   You must connect the two AggregateControl nodes in parallel, not in sequence. This means that you must connect both to the Filter node (one to the true terminal, one to the false), and both to the downstream nodes that handle the requests for the fan-out. Each input message must pass through only one of the AggregateControl nodes. If you connect the nodes such that a single message is processed by more than one AggregateControl node, duplicate records are created in the database by the AggregateRequest node and subsequent processing results are unpredictable.

Results

The following diagram shows an example fan-out message flow that uses this technique.



Correlating input request and output response aggregation messages
If you want to correlate initial request messages with their combined response messages, you can do so using the ReplyIdentifier in the Properties folder of the response message.

Before you begin

To complete this task, you must have completed the following tasks:

- “Creating the aggregation fan-out flow” on page 1823
- “Creating the aggregation fan-in flow” on page 1826

About this task

In some cases you might want to correlate aggregation request messages with the combined response message produced by your fan-in flow, there are two ways of doing this:

- Store some correlation information in one of the requests sent out as part of the aggregation.
- Send the original request message directly back to the AggregateReply node as one of the aggregation requests. To do this, the CorrelId must be set to the MsgId, and the MQOutput node must have its MessageContext set to 'Pass all'.

Using control messages in aggregation flows
The default behavior is that connections between AggregateControl and AggregateReply nodes for sending control messages are ignored. This configuration optimizes performance and removes the possibility that response messages will be received by the AggregateReply node before the control message.

Before you begin

To complete this task, you must have completed the following tasks:

- "Creating the aggregation fan-out flow" on page 1823
- "Creating the aggregation fan-in flow" on page 1826

About this task

Control messages are not required to make aggregations work correctly. However, you can send control messages in your aggregation flows if you want. To send control messages in a message flow, see "Configuring message flows to send control messages" on page 1835 and "Configuring an integration node environment to send control messages" on page 1836.

Configuring message flows to send control messages

About this task

To configure message flows to send control messages from an AggregateControl node to an AggregateReply node:

Procedure

1. If you have created the fan-out and fan-in flows in a single message flow:
   a) Open the aggregation message flow.
   b) Connect the Control terminal of the AggregateControl node to the Control terminal of the AggregateReply node to make the association.

   This connection is referred to as a direct connection between the two aggregation nodes.
2. If you have created separate fan-out and fan-in message flows:
   a) Open the fan-out message flow.
   b) Configure the AggregateControl node, see "Creating the aggregation fan-out flow" on page 1823.
   c) At this stage, you can configure a Compute node that creates a valid output message that contains the control message. For example, to pass the control message to an MQOutput node, configure the Compute node to add an MQMD to the message and complete the required fields in that header. For example, you can code the following ESQL:

   ```
   SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
   SET OutputRoot.MQMD.Format = MQFMT_STRING;
   ```

   d) Configure an output node that represents the intermediate destination for the control message. For example, to send the control message to an intermediate IBM MQ queue, include an MQOutput

node and identify the target queue in the Basic properties `Queue Manager Name` and `Queue Name`.

e) Connect the Control terminal of the AggregateControl node to the In terminal of the Compute node, and connect the Out terminal of the Compute node to the In terminal of the output node that represents the intermediate destination for the control message.

f) Open the fan-in message flow.

g) Configure one input node to receive the reply messages, see "Creating the aggregation fan-in flow" on page 1826. This input node also receives the control information from the AggregateControl node. For example, set the Basic property `Queue Name` of the MQInput node to receive the response and control message from an intermediate IBM MQ queue.

h) Add a Filter node to your fan-in flow after the input node and before the AggregateReply node, see "Avoiding thread starvation on fan-in flows" on page 1837.

i) Connect the Out terminal of the input node to the In terminal of the Filter node.

j) Connect the Out terminals of the Filter node to the Control terminal and in terminal of the AggregateReply node.

This connection is referred to as an indirect connection between the two aggregation nodes.

Configuring an integration node environment to send control messages

About this task

By default, in WebSphere Message Broker Version 8.0, all connections from the Control terminal of the AggregateRequest node to the AggregateReply node are ignored. For these connections to be active, create the MQSI_AGGR_COMPAT_MODE environment variable in the integration node environment. By default, the environment variable does not exist. The existence of the environment variable means that connections from the AggregateControl node are active, regardless of the value to which the environment variable is set.

When the MQSI_AGGR_COMPAT_MODE environment variable has not been created, the default behavior for aggregation fan-out flows is used. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, this connection is ignored and no control message is sent.

If the MQSI_AGGR_COMPAT_MODE environment variable is created, the default behavior for aggregation fan-out flows is not used, allowing you to send control messages from the AggregateControl node to the AggregateReply node. If the Control terminal of the AggregateControl node is connected, either directly or indirectly, to the In terminal of the AggregateReply node, see "Creating the aggregation fan-out flow" on page 1823, this connection is recognized and a control message is sent. Be aware that this configuration is not the optimal configuration and might affect performance.

To create the MQSI_AGGR_COMPAT_MODE variable to support connections between AggregateControl and AggregateReply nodes to be recognized:

- Windows On Windows:
  1. Open System Properties by clicking **Start** > **Control Panel** > **System**.
  2. Click the **Advanced** tab.
  3. Click **Environment Variables**.
  4. In the **System variables** pane, click **New**.
  5. Under **`Variable name`** type `MQSI_AGGR_COMPAT_MODE`.
  6. (Optional) You can type in the **`Variable value`** or leave it blank.
  7. For the environment variable to take effect, restart the computer.

- Linux UNIX z/OS On Linux, UNIX and z/OS:
  1. Edit the profile of the integration node userid and include the following code:

     ```
     export MQSI_AGGR_COMPAT_MODE=
     ```

  2. Reload the profile.
  3. Restart the integration node.

Avoiding thread starvation on fan-in flows
Follow this guidance to avoid thread starvation on fan-in flows if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

About this task

By not connecting the Control terminal, you can overcome the issues that are discussed here. For further information about connecting the Control terminal of the AggregateControl node, see “Using control messages in aggregation flows” on page 1835.

The Aggregate Reply node has two input terminals: In and Control. The use of the Control terminal is optional. If you use both of these terminals, the MQInput nodes that supply the two terminals must not use threads from the message flow additional instance pool. If the nodes do use these threads, they compete for resources, and the Control terminal's MQInput node typically takes all available threads because it is activated before the In terminal.

Configure each MQInput node to use additional instances that are defined on the node, not at the message flow level.

Handling exceptions in aggregation flows
When you use aggregation flows, exceptions might occur.

Before you begin

Complete the following tasks:

- “Creating the aggregation fan-out flow” on page 1823
- “Creating the aggregation fan-in flow” on page 1826

Dealing with exceptions

About this task

If an error is detected downstream of an AggregateReply node, the integration node issues an exception. Another node in the message flow might also issue an exception using the ESQL THROW statement. In either case, when an exception occurs, it is caught in one of two places:

- The input node on which the replies arrive
- The AggregateReply node

The following table lists events and describes what happens to an exception that occurs downstream of the AggregateReply node.

Event	Message propagated	Output terminal	Exception caught at
An expected reply arrives at the input node and is passed to the In terminal of the AggregateReply node. The reply is the last one that is needed to make an aggregation complete.	An aggregated reply message that contains all the replies	Out	Input node

Event	Message propagated	Output terminal	Exception caught at
An unexpected reply arrives at the input node and is passed to the AggregateReply node. The reply is not recognized as a valid reply, and the `Unknown Message Timeout` property is set to `0`.	Message received	Unknown	Input node
A timeout occurs because all the replies for an aggregation have not yet arrived.	An aggregated reply message that contains all the replies that have been received	Timeout	AggregateReply node
An unknown timeout occurs because a retained message is not identified as a valid reply.	Retained message	Unknown	AggregateReply node
An aggregation is discovered to be complete at some time other than when the last reply arrived.	An aggregated reply message that contains all the replies	Out	AggregateReply node

To handle errors that occur in aggregation flows, you must catch these exceptions at all instances of each of these nodes in the message flow.

Procedure

1. Switch to the Integration Development perspective.
2. Open the message flow with which you want to work.
3. To handle these exceptions yourself, connect the Catch terminal of each input and AggregateReply node to a sequence of nodes that handles the error that has occurred.

   For a unified approach to error handling, connect the Catch terminals of all these nodes to a single sequence of nodes, or create a subflow that handles errors in a single consistent manner, and connect that subflow to each Catch terminal.
4. If you want the integration node to handle these exceptions using default error handling, do not connect the Catch terminals of these nodes.

Results

If you connect the Catch terminal of the AggregateReply node, and want to send the message that is propagated through this terminal to a destination from which it can be retrieved for later processing, include a Compute node in the catch flow to provide any transport-specific processing. For example, you must add an MQMD header if you want to put the message to an IBM MQ queue from an MQOutput node.

The following ESQL example shows you how to add an MQMD header and pass on the replies that are received by the AggregateReply node:

```
-- Add MQMD
SET OutputRoot.MQMD.Version = 2;
.
-- Include consolidated replies in the output message
SET OutputRoot.XMLNS.Data.Parsed = InputRoot.ComIbmAggregateReplyBody;
.
```

To propagate the information about the exception in the output message, set the `Compute mode` property of the Compute node to a value that includes `Exception`.

Exceptions when dealing with unknown and timeout messages
When timeout messages or unknown messages from unknown timeout processing are produced from an AggregateReply node, they originate from an internal queue and not from an MQInput node. This behavior affects how the error handling should be performed.

If a message that is sent down the timeout thread causes an exception, the message rolls back to the AggregateReply node and is sent to the Catch terminal. If this terminal is unattached or an exception occurs while processing the message, the timeout is rolled back onto the internal queue and is reprocessed. Potentially, this behavior can lead to an infinite loop, which can be stopped by deploying a version of the message flow that fixes the problem.

To avoid this infinite loop, take the following actions.

- Connect the Catch terminal to a set of nodes that handle errors.
- Ensure that the error-handling nodes cannot throw an exception by ensuring that they perform very simple operations; for example, converting the message to a BLOB, then writing it to a queue, or adding extra TryCatch nodes.

The failure terminal of the AggregateReply node is not used currently and messages are not passed to this terminal.

Configuring the storage of events for aggregation nodes
You can use an Aggregation policy to control the storage of events for AggregateControl and AggregateReply nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using aggregation on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the aggregation nodes. If you are using aggregation on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see “Using a remote default queue manager” on page 750 and “Configuring an integration server to use a remote default queue manager” on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing aggregation nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in “Creating the default system queues on an IBM MQ queue manager” on page 99.

By default, the storage queues used by all aggregation nodes are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.UNKNOWN
- SYSTEM.BROKER.AGGR.TIMEOUT

However, you can control the queues that are used by different aggregation nodes by creating alternative queues containing a *QueuePrefix*, and using an Aggregation policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry time of an aggregation:

Procedure

1. Create the storage queues to be used by the aggregation nodes.

   The following queues are required:

   - SYSTEM.BROKER.AGGR.*QueuePrefix*.CONTROL
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REPLY
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.REQUEST
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.UNKNOWN
   - SYSTEM.BROKER.AGGR.*QueuePrefix*.TIMEOUT

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET`.`1 are valid queue prefixes, but `.`SET1 and SET1`.` are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Create an Aggregation policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific aggregation or with all aggregations in an integration server. If the policy is to be used with a specific aggregation, create the policy with the same name as the name that you specify in the `Aggregate name` property on the AggregateControl and AggregateReply nodes.

      To specify a default Aggregation policy for all message flows that are deployed to an integration server, set the **`Aggregation`** property in the `server.conf.yaml` file to the name of an Aggregation policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*
   b) Set the **`Queue prefix`** property of the Aggregation policy to the required value (see Aggregation policy).
   c) Optional: Set the **`Timeout`** property of the Aggregation policy to control the expiry time of an aggregation.

   If you delete the Aggregation policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.
3. In the AggregateControl and AggregateReply nodes, ensure that the name of the Aggregation policy is the same as the name that is specified in the `Aggregate name` property on the **Basic** tab; for example, `myAggregation`.

   If there is no Aggregation policy with the same name as the `Aggregate name` node property value, and if there is a default Aggregation policy specified in the `server.conf.yaml` file, that Aggregation policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Using message collections

A message collection is a single message that contains multiple messages derived from one or more sources. You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes. Information about the state of in-flight messages is held on system queues that are controlled by IBM MQ.

About this task

If you want to use message collections on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node. The system queues that hold the state information are owned by the queue manager that is specified on the integration server. You associate a queue manager with the integration server by specifying the queue manager name on the **defaultQueueManager** property in the `server.conf.yaml` configuration file; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

You must create the system queues by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

If you want to use message collections with an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

The topics in this section describe how message collection works, and explain how you can configure your message flow to process message collections.

Message collections

A message collection is a single message that contains multiple messages derived from one or more sources.

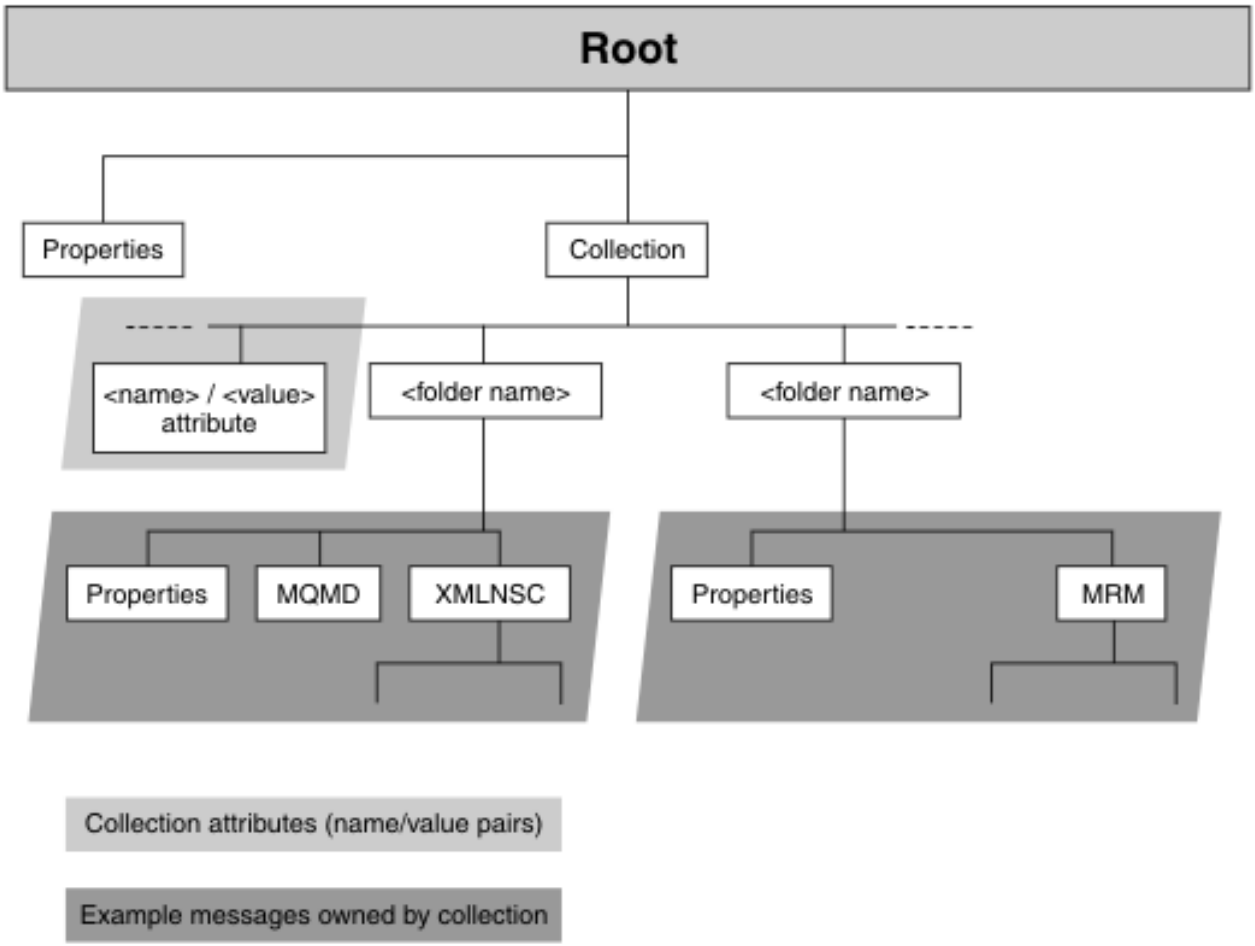
You can use a Collector node to group together messages from one or more sources into a message collection, so that they can be processed together by downstream nodes. You can also manually build a message collection using a Compute node.

Structure of a message collection

A message collection tree contains sub-trees that hold the content of the individual messages received by the Collector node. The message assembly that is propagated from the Collector node to other nodes in your message flow contains the following four components:

- Message (including transport headers)
- Local environment
- Global environment
- Exception list

The following figure shows an example of the message tree structure of a message collection.



The message collection in this example contains two messages, one received from IBM MQ, and one from a file input source.

A message collection has a Properties header and a single folder element named Collection. A message collection can also have zero or more attributes that are name/value pairs; the name of an attribute must be unique within a message collection. These are shown as `<name> / <value>` in the figure. A standard attribute for the message collection is an attribute called *`CollectionName`*. If you use a Collector node to generate a message collection, the value for the collection name is generated based on the values you configure in the node. The collection name attribute is not compulsory.

Within the Collection folder in the message collection tree structure are folders, shown as `<folder name>` in the diagram. These folders contain the message tree of each message added to the message collection. Each of these folders has a name, but this name does not have to be unique within the message collection. The value for the `<folder name>` is derived from the source of the input message.

Nested message collections are not permitted. You cannot therefore use a message collection as a source message for another message collection. For example, If you attempt to pass a message collection to a input terminal on a Collector node, an error is generated.

The LocalEnvironment, Environment and ExceptionList trees are not included in the structure, but are instead carried separately as a part of the message assembly. There is no concept of a LocalEnvironment associated with each message in a message collection.

Generating a message collection using a Collector node

You can use the Collector node to make multiple synchronous or asynchronous requests in parallel. The results of these requests can be joined together downstream if required. This is different from the behavior of the aggregation nodes where there is a fixed pattern of request/response and where reply

messages are grouped by request id. In contrast, the collector node does not need an initial fan-out stage and can group together unrelated input messages by correlating their content. You can configure dynamic input terminals on a Collector node to receive messages from different sources. You can also configure properties on the Collector node, known as event handlers, to determine how messages are added to a message collection, and when a message collection is complete.

Processing a message collection

A message collection is supported by the following nodes only:

- Compute
- JavaCompute
- CICSRequest

Errors are generated by other nodes if they receive a message collection.

You can use ESQL or XPath expressions to access the content of messages in a message collection by referencing the folder names preceded by `InputRoot.Collection`. To access the contents of a message in a message collection using ESQL you can use code similar to the following ESQL:

```
InputRoot.Collection.folder1.XMLNSC
```

In XPath, the root element is the body of the message. The root element for a message collection is the Collection element. Therefore, to access the contents of a message in a message collection using XPath, you must use an expression similar to the following XPath:

```
/folder1/XMLNSC
```

Examples of XPath expressions that you can use to access the message collection are:

- `/*`: returns a list of all the messages in the message collection.
- `/@*`: returns a list of all the attributes of the message collection.
- `/@Name`: returns the value of the attribute `Name`.

You might not be able to determine the order of the messages within a message collection. If you generate a message collection using the Collector node, the messages are arranged in the same order as the messages arrived at the node.

Creating a message collection by using ESQL

A message collection can be constructed by using ESQL. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

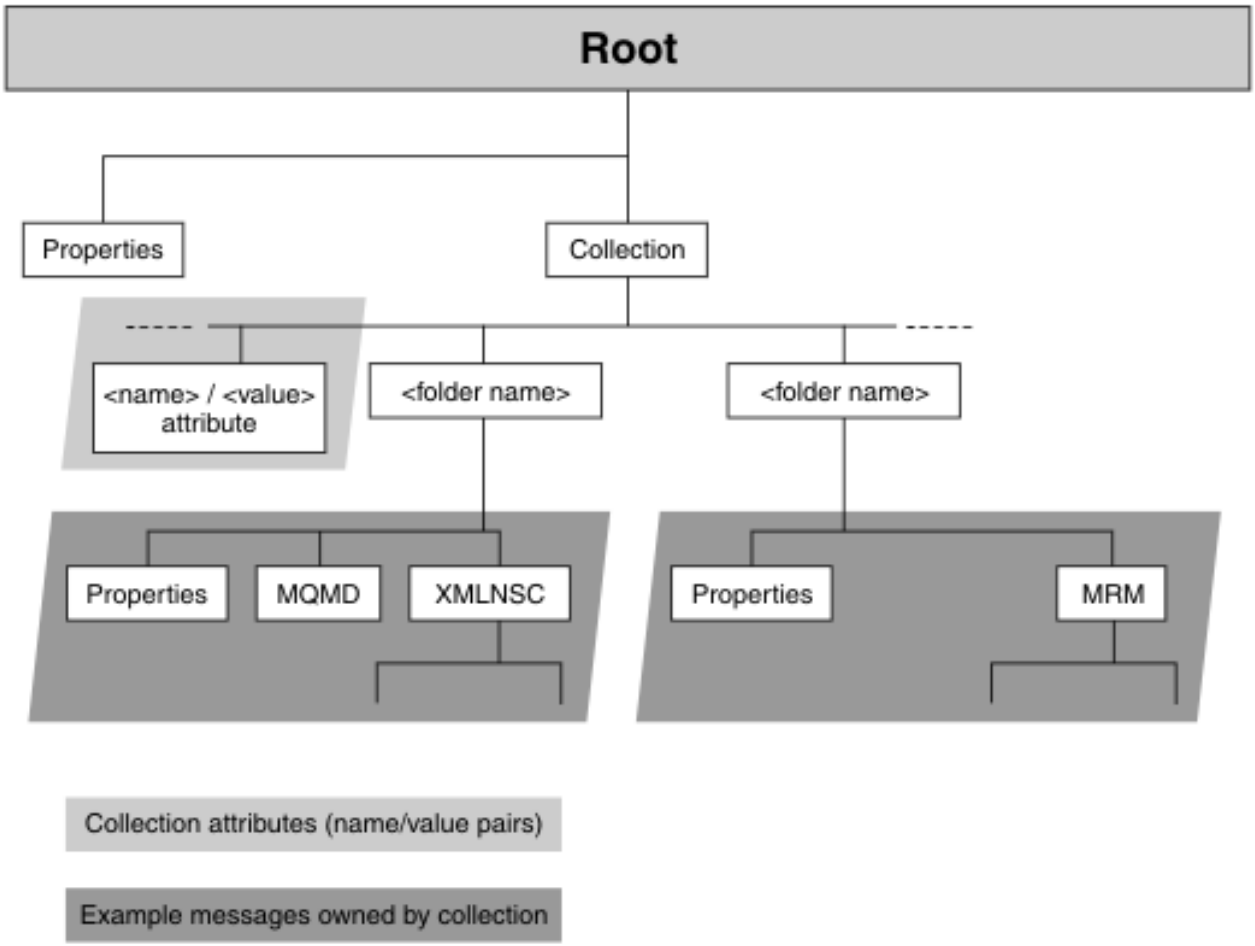
Before you begin

- Create an application, library, or integration project.
- For background information, read “Message collections” on page 1841.

About this task

A message collection is a message that consists of a `Properties` header and a single domain element named `Collection`. The `Collection` folder contains a number of child messages, each of which can contain a `Properties` folder, a number of headers (such as MQMD), and a body. A message collection can also have zero or more attributes that are name/value pairs. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called `CollectionName`.

The following figure shows an example of a message collection structure.



You can create a message collection by using ESQL to group messages together for parsing, or create a message collection that must be constructed to represent a particular data structure, such as a CICS channel data structure.

To configure a message collection by using ESQL, complete the following steps:

Procedure

1. Create a `Properties` folder for the collection by using the following ESQL statement:

```
CREATE FIRSTCHILD of OutputRoot domain 'Properties' NAME 'Properties';
```

2. Create the `Collection` domain element by using the following statement:

```
CREATE LASTCHILD OF OutputRoot DOMAIN 'Collection';
```

   As with message folders, the domain element is always the last child of the message.

3. Use the following statement to set an attribute in the collection called `CollectionName`:

```
SET OutputRoot.Collection.CollectionName = 'myCollectionName';
```

4. The following ESQL shows an example procedure to create a message within the collection:

```
SET OutputRoot.Collection.foldername.Properties.MessageSet     = set;
SET OutputRoot.Collection.foldername.Properties.MessageType    = type;
SET OutputRoot.Collection.foldername.Properties.MessageFormat  = format;
SET OutputRoot.Collection.foldername.Properties.Encoding       = encoding;
SET OutputRoot.Collection.foldername.Properties.CodedCharSetId = ccsid;
```

```
SET OutputRoot.Collection.foldername.domain.content=some data;
```

Creating a message collection by using Java

A message collection can be constructed by using Java and the MbMessageCollection class. Using a message collection is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure.

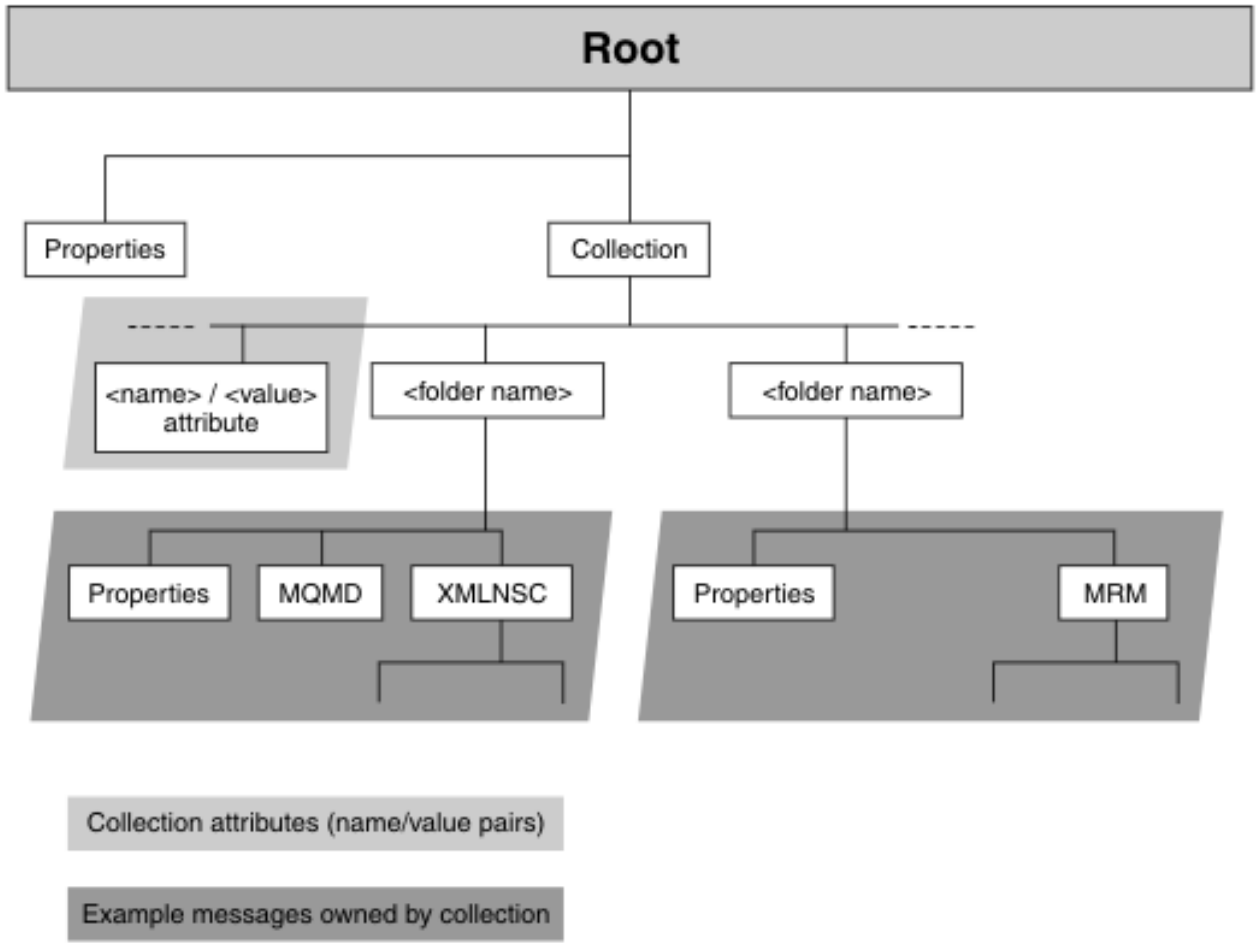
Before you begin

- For background information, read "Message collections" on page 1841.

About this task

A message collection is a message that consists of a `Properties` header and a single domain element named `Collection`. The `Collection` folder contains a number of child messages, each of which can contain a `Properties` folder, a number of headers (such as MQMD), and a body. A message collection can also have zero or more attributes that are name/value pairs. The name of an attribute must be unique within a message collection. A standard attribute for the message collection is an attribute called *`CollectionName`*.

The following figure shows an example of a message collection structure.



You can create a message collection by using Java, and the MbMessageCollection class, to group messages together for parsing, or create a message collection that must be constructed to represent a particular data structure, such as a CICS channel data structure.

To configure a message collection by using Java, complete the following steps:

Procedure

1. Create a new message by using the following example:

```
// create new message
MbMessageCollection outMessage = new MbMessageCollection();
MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,
 outMessage);
```

2. Create a `Properties` folder for the collection by using the following example:

```
// create top level Properties folder and data
MbElement omroot = outMessage.getRootElement();
MbElement properties = omroot.createElementAsFirstChild("Properties");
MbElement property1 = properties.createElementAsLastChild(
 MbElement.TYPE_NAME_VALUE, "myProperty1", "propertyData1");
MbElement property2 = properties.createElementAsLastChild(
 MbElement.TYPE_NAME_VALUE, "myProperty2", "propertyData2");
```

3. Create the name value pairs by using the following example:

```
// create collection attributes (name/value pairs)
MbElement cn  = outMessage.createNameValue("CollectionName", "myCollectionName");
MbElement nv1 = outMessage.createNameValue("NAME1", "Value1");
MbElement nv2 = outMessage.createNameValue("NAME2", 12345);
```

   As with message folders, the domain element is always the last child of the message property.

4. The following example shows the procedure to create a message within the collection. Steps one, two, and three are repeated.

```
public void evaluate(MbMessageAssembly inAssembly) throws MbException {
 MbOutputTerminal out = getOutputTerminal("out");

 // create new message
 MbMessageCollection outMessage = new MbMessageCollection();
 MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,
  outMessage);

 // create top level Properties folder and data
 MbElement omroot = outMessage.getRootElement();
 MbElement properties = omroot.createElementAsFirstChild("Properties");
 MbElement property1 = properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myProperty1", "propertyData1");
 MbElement property2 = properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myProperty2", "propertyData2");

 // create collection attributes (name/value pairs)
 MbElement cn  = outMessage.createNameValue("CollectionName", "myCollectionName");
 MbElement nv1 = outMessage.createNameValue("NAME1", "Value1");
 MbElement nv2 = outMessage.createNameValue("NAME2", 12345);

 // create folder 1
 MbElement folder1 = outMessage.createFolder("folder1");

 // create properties for folder 1
 MbElement folder1properties = folder1.createElementAsFirstChild("Properties");
 MbElement folder1property1 = folder1properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder1Property1", "folder1propertyData1");
 MbElement folder1property2 = folder1properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder1Property2", "folder1propertyData2");

 // create body of folder 1
 MbElement mrm = folder1.createElementAsLastChild("MRM");

 // create message domain element of folder 1
 MbElement msg = mrm.createElementAsLastChild(MbElement.TYPE_NAME,
  "msg", null);

 // create data within the message body for folder 1
 MbElement data = msg.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "data", "myData");

 // create folder 2
 MbElement folder2 = outMessage.createFolder("Folder2");
```

```
 // create properties for folder 2
 MbElement folder2properties = folder2.createElementAsFirstChild("Properties");
 MbElement folder2property1 = folder2properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder2Property1", "folder2propertyData1");
 MbElement folder2property2 = folder2properties.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myFolder2Property2", "folder2propertyData2");

 // create body of folder 2
 MbElement xmlnsc = folder2.createElementAsLastChild("XMLNSC");

 // create message domain element of folder 2
 MbElement msg2 = xmlnsc.createElementAsLastChild(
  MbElement.TYPE_NAME, "msg2", null);

 // create data within the message body for folder 2
 MbElement data2 = msg2.createElementAsLastChild(
  MbElement.TYPE_NAME_VALUE, "myData2", "myXMLData");

 try {
  out.propagate(outAssembly);
 } finally {
  // clear the outMessage even if there's an exception
  outMessage.clearMessage();
 }
}
```

Results

The following example output from a Trace node shows the message structure built by this Java code:

```
TraceOutput:  Root ( ['MQROOT' : 0xee3a90]
 (0x01000000:Name):Properties = ( ['MQPROPERTYPARSER' : 0xae4370]
  (0x03000000:NameValue):myProperty1 = 'propertyData1' (CHARACTER)
  (0x03000000:NameValue):myProperty2 = 'propertyData2' (CHARACTER)
 )
 (0x01000000:Name):Collection = ( ['COLLECTION' : 0x58d0b08]
  (0x03000000:NameValue):CollectionName = 'myCollectionName' (CHARACTER)
  (0x03000000:NameValue):NAME1          = 'Value1' (CHARACTER)
  (0x03000000:NameValue):NAME2          = 12345 (INTEGER)
  (0x01000000:Name     ):Folder1     = ( ['COLLECTIONFOLDER' : 0xee42e8]
   (0x01000000:Name ):Properties = ( ['MQPROPERTYPARSER' : 0xae39e8]
    (0x03000000:NameValue):myFolder1Property1 = 'folder1propertyData1' (CHARACTER)
    (0x03000000:NameValue):myFolder1Property2 = 'folder1propertyData2' (CHARACTER)
   )
   (0x01000021:Name+):MRM        = ( ['mrm' : 0xdce588]
    (0x01000000:Name):msg = (
     (0x03000000:NameValue):data = 'myData' (CHARACTER)
    )
   )
  )
  (0x01000000:Name     ):Folder2     = ( ['COLLECTIONFOLDER' : 0xee3d58]
   (0x01000000:Name  ):Properties = ( ['MQPROPERTYPARSER' : 0xae4cf8]
    (0x03000000:NameValue):myFolder2Property1 = 'folder2propertyData1' (CHARACTER)
    (0x03000000:NameValue):myFolder2Property2 = 'folder2propertyData2' (CHARACTER)
   )
   (0x01000000:Folder):XMLNSC     = ( ['xmlnsc' : 0xee2188]
    (0x01000000:Folder):msg2 = (
     (0x03000000:PCDataField):myData2 = 'myXMLData' (CHARACTER)
    )
   )
  )
 )
)
```

Creating a flow that uses message collections

Use a Collector node in your message flow to group messages from one or more sources into a message collection. You can add dynamic input terminals to your Collector node for each message source that you want to configure for your message flow.

Before you begin

- For background information, read “Message collections” on page 1841.

- If you are using message collections on an integration server that is managed by an integration node, ensure that you have installed IBM MQ. Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ, and you must install IBM MQ on the same computer as your integration node if you want to use the capabilities provided by the aggregation nodes. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

  If you are using message collections on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

About this task

You can also use a Compute node to create a message collection by using ESQL, which is useful if messages must be grouped together for parsing, or if the message collection must be constructed to represent a particular data structure, such as a CICS Transaction Server for z/OS channel data structure. For more information about using ESQL to create a message collection, see "Creating a message collection by using ESQL" on page 1843.

To create a message flow to generate and process message collections, complete the following steps:

Procedure

1. Create a message flow.
2. Add the input nodes in the editor view.

   The input nodes receive the messages from which message collections are generated. You can use any of the built-in nodes, or user-defined input nodes. Configure and connect them as described.

   a) Add an input node for each source of input messages for your message flow, for example an MQInput node and a JMSInput node.

   b) Select each input node in turn to display its properties in the Properties view.

   c) Specify the source of input messages for each node. For example, specify the name of an IBM MQ queue in the Basic property `Queue Name` from which the MQInput node retrieves messages.

   d) Optional: Specify values for any other properties that you want to configure for each node.
3. Add the Collector node in the editor view.

   The Collector node receives messages from input nodes or other nodes in the message flow. You must add a dynamic input terminal to the Collector node for each input message source before you can connect the input nodes or any upstream nodes to the Collector node. Configure and connect them as described.

   a) Add a Collector node to your message flow.

   b) Right-click the Collector node and click **Add Input Terminal** to add a new dynamic input terminal to the Collector node. Add a new input terminal for each input source that you plan to add to your message flow; for more information about adding dynamic input see "Adding an input terminal to a Collector node for each input source" on page 1850.

   c) Connect the out terminal of each input node to a different dynamic input terminal of the Collector node. This represents the simplest configuration; if appropriate, you can include other nodes between the input node and the Collector node. For example, you might want to store the request for audit purposes (in a Database node), or add a unique identifier to the message (in a Compute node).

4. Add processing nodes to your message flow.

   You can process message collections from a Collector node by using the following nodes only:

   - Compute
   - JavaCompute
   - .NETCompute

   You must connect a Compute, JavaCompute, or .NETCompute node to the Collector node in your message flow. Use these nodes to process the message collection and propagate other messages. You can use ESQL or XPATH to access the contents of the individual messages in the message collection for processing. To process a message collection:

   a) Add a Compute, JavaCompute, or .NETCompute node to your message flow.

   b) Add code to your message flow node to create single output messages from the message collection.

   c) Optional: Specify values for any other properties that you want to configure for this processing node.

   d) Connect the out terminal of the processing node to the in terminal of an output node or other processing node.

   e) Optional: Add other message flow nodes to your message flow for further processing.

5. Include one or more output nodes for your message flow. These can be any of the built-in nodes, or a user-defined output node. An output node cannot process a message collection, therefore ensure that you connect the output node to a processing node that propagates single output messages. To configure an output node:

   a) Select each output node in turn to display its properties in the Properties view.

   b) Specify the destination properties for each node. For example, specify the name of an IBM MQ queue in the Basic property `Queue Name` to which the MQOutput node sends messages.

   c) Optional: Specify values for any other properties that you want to configure for each node.

6. Include processing for handling errors and expired message collections:

   a) Optional: Add processing nodes to your message flow to handle expired message collections. Connect these nodes to the Expire terminal of the Collector node.

   b) Optional: Add processing or error handling nodes to handle any exceptions in your message flow. Connect these nodes to the Catch terminal of the Collector node

   If an error is detected downstream of the Collector node, the integration node throws an exception. The message collection is propagated to the Catch terminal of the Collector node. Connect the Catch terminal to a sequence of nodes that handles the errors, to avoid losing any data, and ensure that no further exceptions can be generated during error processing. The message flow node that is connected to the Catch terminal must be a Compute, JavaCompute, or .NETCompute node to handle the message collection.

7. Press Ctrl-S or click **File** > **Save *name*** (where *name* is the name of this message flow) to save the message flow and validate its configuration.

Results

To control when complete message collections are propagated, you must add additional message flow nodes to your message flow. For more information, see “Using control messages with the Collector node” on page 1856.

What to do next

Next: Configure the Collector node.

Configuring the Collector node

You can configure the Collector node to determine how messages are added to message collections. You can also use properties on the Collector node to control when message collections are propagated.

Before you begin

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

Use the following topics to configure the Collector node:

Adding an input terminal to a Collector node for each input source

Add new dynamic input terminals to the Collector node for all of the sources of messages for your message collections.

Before you begin

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

To add a dynamic input terminal to the Collector node for each message source, complete the following steps.

Procedure

1. Right-click the Collector node and select **Add Input Terminal**.
2. In the dialog box that is displayed, enter a name of your choice for the terminal, and click **OK**.
   The name that you give to the input terminal is used as the folder name in the message collection.
3. Repeat steps "1" on page 1850 and "2" on page 1850 to add further input terminals.

What to do next

When you have created all the required input terminals on the Collector node, you can set the event handler properties. For more information see, "Setting event handler properties" on page 1850.

Setting event handler properties

You can configure event handler properties for each dynamic input terminal on a Collector node. These event handler properties determine how the messages received by each terminal are added to message collections.

Before you begin

To complete this task, you must have completed the following tasks:

- "Creating a flow that uses message collections" on page 1847

- "Adding an input terminal to a Collector node for each input source" on page 1850

About this task

You can use one or more of the event handler properties to control the way that messages are added to message collections for each input terminal that you added to the Collector node. Incomplete message collections are stored on an IBM MQ queue. The message collections are stored in the order that they are generated by the Collector node (first in, first out). Each message collection has an event handler instance for each of the input terminals. The event handler determines whether an incoming message on that terminal is added to a message collection. The event handler instance maintains information about the state of the collection, the number of messages received, the timer, and the correlation string. When a new message is received on an input terminal, it is offered to the event handler for each message collection waiting on the queue in turn. When the message is accepted by one of the event handlers, it is added to the message collection. The accepted message is not offered to any other message collections. If all the event handlers reject the message, it is added to a new message collection, which is added to the end of the queue.

The first message accepted into a collection determines the correlation string for that message collection, if it is configured. Subsequent messages offered to that message collection are only accepted if their correlation string matches that of the collection. The first message accepted by each event handler starts the timeout timer, if it is configured. Each message accepted by each event handler increments the quantity count. An event handler becomes satisfied when the number of messages accepted equals the configured quantity, or when the timeout value is reached. When an event handler is satisfied, the event handler does not accept any more messages. A message collection is complete only when all of the message collection's event handlers are satisfied. The message collection is then ready for propagation.

You can configure the event handler properties by using the **Collection Definition** table, on the **Basic** tab of the Properties view.

To configure the event handler properties on the Collector node:

Procedure

1. Open the message flow with the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. Use the following instructions to configure the event handler properties that you want to set for each input terminal:
   - If you want to add a set number of messages to each message collection from one or more of the terminals, you must enter a value for `Quantity` in the Collection Definition table. This value is used to specify the number of messages that each configured input terminal accepts to complete a collection. For example, if you have set `Quantity` to wait for 2 messages on three of the input terminals, the message collection is not complete until 2 messages have been received on each of the three input terminals. The complete message collection contains 6 messages, 2 from each of the three terminals. As soon as more than 2 messages are received on one of the input terminals, the next message is added to a new message collection.
     a. In the Collection Definition table, click the row for the selected input terminal within the `Quantity` column.
     b. Enter a value for the number of input messages that you want to add to a message collection. If you select `Zero` or choose not to set this property, there is no limit to the number of messages accepted. In this case the value set on the `Timeout` property must be greater than zero. If you accept the default value of 1; only one message from the selected terminal is added to a collection.

     You must enter a value for `Quantity` if `Timeout` is not set.
   - If you want to collect messages for a set amount of time before the message collection is propagated you must enter a value for `Timeout`. This value is used to specify the maximum

time in seconds for which the selected input terminal accepts messages before completing a message collection. The timeout interval starts when the first message has arrived at the selected terminal. Any subsequent messages are added to the same message collection. When the timeout interval ends, no more messages are added to the message collection from this terminal. When the conditions on all the terminals are satisfied, the message collection is ready for propagation. When the next message reaches the selected input terminal, a new message collection is created and the timeout interval starts again. If a timeout is set on multiple input terminals, each terminal collects messages for the configured amount of time. During the timeout messages from all of the terminals are added to the same message collection.

a. In the Collection Definition table, click the row for the selected input terminal within the `Timeout` column.

b. Enter a value for the length of time in seconds that you want to wait to add messages to a message collection. For example, to wait for messages to add to a message collection for an hour, enter a value of `3600`. If you accept the default value `Zero`, timeout is not enabled and there is no limit on the time to wait for messages. In this case the value set on the `Quantity` property must be greater than zero.

You must enter a value for `Timeout` if `Quantity` is not set.

- If you want to add messages to different message collections based on the content of the message you must enter an XPath value for the `Correlation path`. This value is used to specify the path in the incoming message from which to extract the correlation string. The correlation string is the value that is extracted by the correlation path. If a correlation pattern is specified, the correlation string is matched against the correlation pattern. Messages are only accepted into a message collection with the same correlation string. If you specify an asterisk (`*`) in the name of the message collection, it is replaced by the correlation string.

  a. In the Collection Definition table, click the row for the selected input terminal within the `Correlation path` column.

  b. Either select a predefined correlation path from the list, or enter your own correlation path using XPath. The correlation path must begin with a correlation name, which can be followed by zero or more path fields. For example, in the following message the correlation string is `xxx` in the name field:

```
<library>
 <name>xxx</name>
 <more>
  ...
 </more>
</library>
```

  In this example, the correlation path using XPath is `$Body/library/name`.

  The variables `$Root`, `$LocalEnvironment`, and `$Environment` are available to allow the path to start at the roots of the message, local environment, environment trees, and message body.

  If the correlation path evaluates to an empty string the unmatched message is added to a default unnamed message collection.

  If you define a value for `Correlation path`, you can optionally configure a `Correlation pattern`.

- If you want to match a substring of the message content from the `Correlation path`, you can define a pattern to match in the message by using `Correlation pattern`. The `Correlation pattern` contains a single wildcard character and optional text. The correlation string, used for the name of the message collection, is the part of the substring that matches the wildcard. For example, if the correlation path contains the filename `part1.dat` in a file header, and the correlation pattern is specified as `*.dat`, the correlation string is `part1`. If this property is set, only messages that have the same correlation string are added to the same message collection. The first

message added to a message collection determines the correlation string that must be matched by all other messages in that message collection.

a. In the Collection Definition table, click the row for the selected input terminal within the `Correlation pattern` column.

b. Enter a value for the correlation pattern. The `Correlation pattern` must contain a single wildcard character: *. This wildcard character can optionally be surrounded by other text.

If the correlation pattern fails to match the wildcard to a substring, the unmatched message is added to a default unnamed message collection.

5. Repeat step "4" on page 1851 for each of the input terminals that you added to your Collector node.

You can configure different event handlers for different input sources.

What to do next

Note: Ensure that you set the event handler properties across different terminals carefully to match the expected delivery of messages to the terminals on the Collector node.

You can now configure the collection expiry, see "Setting the collection expiry" on page 1853.

Setting the collection expiry

The collection expiry is a property on the Collector node to set a maximum timeout for adding messages to a message collection.

Before you begin

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

When messages are added to a message collection, the incomplete message collection is stored on a queue. If the message collection's event handlers are not satisfied, the incomplete message collection is stored on the queue indefinitely, and not propagated for further processing. If a Collector node has 2 input terminals, and one of the terminals stops receiving messages, for example if the source application is not running, there is the potential for the queue of incomplete message collections to grow indefinitely. To ensure that these incomplete message collections are released after an appropriate amount of time, configure the *`Collection Expiry`* property. You can configure this timeout, as a value in seconds, in the *`Collection Expiry`* property on the Collector node. The collection expiry timeout starts when the first message is accepted into a message collection. The collection expiry overrides any individual event handler timers. When the collection expiry timeout has passed for a message collection, the incomplete message collection is propagated to the **Expire** terminal. Connect appropriate processing nodes to the **Expire** terminal, to handle any expired message collections in your message flow.

To configure a collection expiry:

Procedure

1. Open the message flow with the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. In `Collection Expiry`, enter a time in seconds for the collection expiry timeout.

What to do next

Next: Configure the collection name.

Setting the collection name
You can set a default name, or use a correlation string, for the name of your message collections, by using the `Collection name` property on the Collector node.

Before you begin

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

Each message collection that is produced by the Collector node has a name. The collection name is the value that is associated with the CollectionName attribute in the message collection tree structure. Each message collection has only one name.

You can either use `Collection name` to set a default name to be used for each message collection, or you can use the event handler properties to create a correlation string to use for the message collection name. You can use the correlation string to generate a unique name for the message collection, based on the content of the input messages. To use the correlation string for the collection name, you must enter the wildcard symbol `*`. If you leave `Collection name` blank, or if it is set to `*` and the value of the correlation string is empty, the CollectionName attribute of the message collection is set to an empty value.

Any `*` characters in the collection name are replaced with the correlation string. The correlation string for each message collection is also copied into the local environment message that is associated with the propagated message collection. The location of the correlation string in the local environment is `Wildcard/WildcardMatch`.

To configure the message collection name:

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Basic** tab.
4. For the `Collection name` property, enter a name for the message collections that are generated by the Collector node.

   If you have set a value for `Correlation path` on your input terminals, you can use the `*` for the `Collection name` property to substitute the correlation string into the collection name value. Leave the collection name blank if you want to set your message collection name to an empty value.

What to do next

Next: "Setting the event coordination property" on page 1854.

Setting the event coordination property
Use the `Event coordination` property for controlling how message collections are propagated from the Collector node.

Before you begin

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

In addition to the dynamic input terminals that you can add to the Collector node, there is a static input terminal called **Control**. The purpose of this terminal is to allow an external resource to trigger the output from the collector node. Details are controlled through the Event coordination property settings.

Incomplete message collections that have exceeded the value for the *Collection expiry* timeout are immediately propagated to the Expire terminal, regardless of how you configure the *Event coordination* property.

To configure *Event coordination*:

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.
4. Set the *Event coordination* property on the Collector node.
   Select from the following options:
   - If you select Disabled, messages to the **Control** terminal are ignored and message collections are propagated when they are complete.
   - If you select All complete collections, complete message collections are held on a queue. When a message is received on the **Control** terminal, all message collections on the queue are propagated to the **Out** terminal.
   - If you select First complete collection, complete message collections are held on a queue. When a message is received on the **Control** terminal, the first message collection on the queue is propagated to the **Out** terminal. If the queue is empty when a message arrives on the **Control** terminal, the next message collection that is completed is propagated to the **Out** terminal.

Results

You have completed configuration of the Collector node.

What to do next

Next: if you have configured your Collector node to use control messages, see "Using control messages with the Collector node" on page 1856.

Setting the persistence mode property

Use the Persistence Mode property to control whether incomplete message collections are stored persistently on the queues of a Collector node.

Before you begin

This topic assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

The storage of incoming messages and the Collector node state is handled internally by using IBM MQ queues. By default, incomplete message collections are stored non-persistently, which means that incomplete message collections persist if you restart your integration node, but not if you restart your queue manager.

You can use the Persistence Mode property on the Collector node to store incomplete message collections on a queue persistently. If you set the Persistence Mode property to Persistent, incomplete message collections are not lost if you restart your queue manager. However, if you do set the property to Persistent, the overall performance of the Collector node might be affected.

To configure the Persistence Mode:

Procedure

1. Open the message flow that contains the Collector node.

2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.
4. Set the `Persistence Mode` property on the Collector node to one of the following values.
   - If you select `Non Persistent`, messages and collection state are stored by the integration node queue manager as non-persistent messages.
   - If you select `Persistent`, messages and collection state are stored by the integration node queue manager as persistent messages.

Results

You have completed configuration of the Collector node.

Setting the Policy property on the Collector node

Use the `Policy` property on the Collector node to specify a Collector policy that sets the values of properties at run time.

Before you begin

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

You can use the `Policy` property on the Collector node to specify the name of a Collector policy to be used by the node. The Collector policy has the following properties:

Queue prefix
The identifier used to specify the names of queues in which events and collections are stored.

Collection expiry (seconds)
The expiry time (in seconds) of a collection. This property overrides the value specified by the `Collection expiry` property on the Collector node.

To configure the `Policy` property, complete the following steps.

Procedure

1. Open the message flow that contains the Collector node.
2. Right-click the Collector node and select **Properties**.
3. Click the **Advanced** tab.
4. Specify the name of the Collector policy in the `Policy` field.

What to do next

Deploy the Collector policy to an integration server before you deploy the message flow that contains the Collector node.

Using control messages with the Collector node

You can send control messages to the Collector node in order to control how complete message collections are propagated to other nodes in your message flow.

Before you begin

This task assumes that you have already created a message flow that contains a Collector node. For more information, see "Creating a flow that uses message collections" on page 1847.

About this task

You can control when complete message collections are propagated to other nodes for processing, using messages sent to the **Control** terminal. The exact behavior depends on the settings that you have chosen for the *Event coordination* property on the Collector node. If you want to use control messages to propagate completed messages collections you must set the *Event coordination* property to one of the following values:

- `All complete collections`
- `First complete collection`

In these cases, the complete message collections are held on a queue until a control message is received. If you set *Event coordination* to `All complete collections`, all the message collections held on the queue are propagated to the **Out** terminal. If you set *Event coordination* to `First complete collection`, only the first message collection on the queue is propagated to the **Out** terminal. If there are no complete message collections on the queue, the next message collection to complete is immediately propagated to the **Out** terminal.

Incomplete message collections that have exceeded the value for *Collection expiry* are immediately propagated to the **Expire** terminal regardless of the setting of *Event coordination*.

If you want to propagate any complete message collections after a set amount of time for further processing, connect a TimeoutNotification node to the **Control** terminal of the Collector node. You can use the TimeoutNotification node to send a control message to propagate the message collections to ensure that messages are processing within a reasonable time, or to schedule processing tasks.

For more information about driving a message flow using the TimeoutNotification node, see "Automatically generating messages to drive a flow" on page 1880.

Alternatively, you can propagate complete message collections using a message from another application or message flow by connecting an input node to the **Control** terminal of the Collector node.

You can send any message to the **Control** terminal of the Collector node. The message received on the **Control** terminal is not examined by the integration node and is discarded on receipt.

Configuring the storage of events for Collector nodes

You can use a Collector policy to control the storage of events for Collector nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using message collections on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the Collector node. If you are using message collections on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow that contains Collector nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queues used by all Collector nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Resequence node.

However, you can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the expiry for the collection:

Procedure

1. Create the storage queues to be used by the Collector node.

   The following queues are required:

   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET.1 are valid queue prefixes, but .SET1 and SET1. are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.
2. Create a Collector policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific collection or with all collections in an integration server. If you are creating a policy to be used with a specific collection, ensure that the name of the policy is the same as the name that you specify in the `Policy` property on the Collector node.

      To specify a default Collector policy for all message flows that are deployed to an integration server, set the **`Collector`** property in the `server.conf.yaml` file to the name of a Collector policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*
   b) Set the **`Queue prefix`** property of the Collector policy to the required value.
   c) Optional: Set the **`Collection expiry`** property of the Collector policy to an appropriate value (for example, 60).

   If you delete a Collector policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.
3. In the Collector node, if the policy is to be used for a specific collection, specify the name of the policy in the `Policy` property on the **Advanced** tab; for example, `myCollectorService`.

   If you do not set the `Policy` property, and if there is a default Collector policy specified in the `server.conf.yaml` file, that policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Using message sequences

You can maintain or change the sequence of messages in a message flow based on the sequence number and group ID contained in each message.

Before you begin

- Read the following concept topics, which contain information that you need to understand before you can use message sequences:
  - "Message sequencing" on page 1860
  - "Sequence groups" on page 1861
  - "Starting a message sequence" on page 1862
  - "Ending a message sequence" on page 1864
  - "Duplicate message processing" on page 1865
- If you are using message sequencing on an integration server that is managed by an integration node, ensure that you have installed IBM MQ. Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ, and you must install IBM MQ on the same computer as your integration node if you want to use the capabilities provided by the Sequence and Resequence nodes. For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

  If you are using message sequencing on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

About this task

Information about the state of in-flight messages is held on system queues that are controlled by IBM MQ, and these queues are owned by the queue manager that is specified on the integration server. You create these system queues by running the **`iib_createqueues`** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

You associate a queue manager with the integration server by specifying the queue manager name on the **`defaultQueueManager`** property in the `server.conf.yaml` configuration file; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

For more information about using IBM MQ with IBM App Connect Enterprise, see "Installing IBM MQ" on page 96.

IBM App Connect Enterprise enables you to configure the sequence of messages in a message flow by using the following nodes:

- node
- node

The following topics describe the tasks involved in configuring message sequences:

- "Adding sequence numbers to messages" on page 1866
- "Reordering messages in a message flow" on page 1868
- "Maintaining the sequential order of messages" on page 1869
- "Handling missing messages" on page 1873
- "Configuring the storage of events for Resequence nodes" on page 232

The following topics provide examples of how to use a Resequence node, and show how the node processes messages in a message flow:

- "Message sequencing scenario 1" on page 1870
- "Message sequencing scenario 2" on page 1871

Message sequencing

Use message sequencing to ensure that messages are delivered to the receiving application in a particular order.

IBM App Connect Enterprise provides support for adding sequence numbers to messages, and for reordering messages in the message flow based on their sequence number. Messages can arrive in any order and you can use the Sequence and Resequence nodes to reorder the messages into the required sequence.

In some applications, the ability to process messages in a specific order is important for maintaining the integrity of the workflow. For example, a series of debits and credits against a bank account must be processed in the order in which they took place, and patient records that are received, processed, and forwarded must be sent on in the order in which they arrived.

Messages arriving in the message flow might or might not contain sequence numbers. If messages without sequence numbers are received from an input source, you can preserve the order in which the messages are received by using a Sequence node to generate a monotonically increasing sequence number for each message in the sequence group. When each message arrives at the Sequence node, the sequence number is incremented and stored with the message in a location specified on the node.

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. If you are using message sequencing on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the Sequence and Resequence nodes. If you are using these nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

The current sequence numbers for each active sequence group are stored on the following IBM MQ queues:

- SYSTEM.BROKER.SEQ.GROUP
- SYSTEM.BROKER.SEQ.NUMBER

For more information, see node.

When the input messages contain sequence numbers, whether they were added by the Sequence node or already defined in an integer field in the message, you can use a Resequence node to change the order of the messages in the message flow.

When messages arrive at the Resequence node, they are held in a storage queue until all previous messages in the sequence have been propagated and committed. When each message becomes the next one in the sequence, it is taken off the queue and propagated down the Out terminal. This sequence of events ensures that messages are kept in the correct order even when message processing fails.

By default, the storage queues used by the Resequence node are:

- SYSTEM.BROKER.EDA.COLLECTIONS
- SYSTEM.BROKER.EDA.EVENTS

However, you can use a Resequence policy to specify alternative queues to be used by the Resequence node (see "Configuring the storage of events for Resequence nodes" on page 232).

You can configure the Resequence node to time out if a message in the sequence fails to arrive in a specified period of time, and you can specify how subsequent messages are processed if a message is missing. For example, you can configure the node so that:

- The message sequence must always be maintained
- Gaps are allowed in the message sequence but all other messages must remain in sequence
- Occasional out-of-sequence messages are allowed

For information about how to configure the Resequence node for these scenarios, see "Handling missing messages" on page 1873.

You can divide messages into sequence groups, which can be processed independently, allowing multiple sequences to be processed at the same time. For more information about sequence groups and duplicate message processing, see "Sequence groups" on page 1861.

For more information about the way in which the beginning and end of sequences are controlled, see "Starting a message sequence" on page 1862.

Sequence groups

Use sequence groups to control the way in which messages are grouped together for processing by Sequence and Resequence nodes.

By default, all messages arriving at the Sequence and Resequence nodes are ordered as part of a single sequence group. However, you can divide messages into multiple sequence groups, based on a sequence group identifier in the message, and order each sequence group independently. For example, you might have a message flow that receives, processes, and forwards patient records. It is important that the order of records is maintained for each individual patient, but ordering between patients is not necessary. In this case, the sequence group identifier would be the name or ID of the patient.

The group to which a message belongs is determined by the group identifier that is specified in the message. You can use the **Path to sequence group identifier** property on the Sequence and Resequence nodes to specify the location of the sequence group identifier in the message.

Messages that have the same group identifier are considered part of the same sequence group. If no sequence group is specified, a single default sequence group is used for all messages. However, if the **Path to sequence group identifier** property specifies a location in the message that does not exist, an error occurs.

Each sequence group can be associated with only one Sequence node. Multiple nodes can have a sequence group with the same name, but each of those sequence groups is associated with only one node and is separate from other groups with the same name on different nodes. For example, SequenceNode1 might have a sequence group called GroupA, and SequenceNode2 might also have a sequence group called GroupA, but they are separate groups.

Although you can reuse sequence groups when they have been closed, there is a risk that two occurrences of the same sequence group could overlap, with unpredictable results.

For example, if you have a sequence group including the numbers 1-10 and the group is used twice in close succession, it is possible for the second occurrence of sequence number *1* to arrive before sequence number *10* of the first occurrence. If this happens, a duplicate message exception occurs. For this reason, it is advisable to use a group name for only one set of sequence numbers, rather than reusing it in a Resequence node. If you do decide to reuse a sequence group, ensure that you reuse it only when you can be certain that the preceding use has been finished for a significant amount of time.

Even if all the numbers have been received by the first occurrence of the group, it can be difficult to know for certain exactly when the group is closed because it closes only when the final message has completely finished processing; this includes any processing that is required downstream of the node. Any message from the second occurrence fails as a duplicate unless all processing is complete. When a sequence group has started overlapping, it is very difficult to recover all the messages (in both uses of the group) in the correct order, although no messages are lost).

Starting a message sequence

The start of a message sequence is determined by the `Start of sequence definition` property on the Resequence and Sequence nodes.

Using a Resequence node

When you use the Resequence node to reorder messages in a message flow, you use the `Start of sequence definition` property on the Resequence node to define how the reordered message sequence will start. You can specify the starting sequence number in one of the following ways:

As a literal number

Select `Literal` to specify a literal sequence number, which can be any positive or negative numeric value in the range -9223372036854775807 to 9223372036854775807. When a message with the specified sequence number arrives, it is identified as the first message in the sequence and the messages are propagated.

Using the smallest number received

Select `Automatic` and specify the length of time (in seconds) during which the node collects messages, before it identifies the message that contains the smallest sequence number. When the smallest number has been determined, that sequence number becomes the first in the message sequence.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`
- `Start of sequence definition` property set to `Automatic` with a value of 5. This value means that, for any new group, the Resequence node collects messages for 5 seconds before determining the starting sequence number.
- `End of sequence definition` property set to `Automatic` with a value of 60. This value means that, for any new group, the Resequence node waits for 60 seconds before determining the ending sequence number.

The following messages are received by the Resequence node:

```
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>b<grp><seq>0</seq></doc>
<doc><grp>b<grp><seq>2</seq></doc>
```

At this point, the automatic period for the start of sequence expires (5 seconds), then the following messages are received:

```
<doc><grp>a<grp><seq>6</seq></doc>
<doc><grp>b<grp><seq>3</seq></doc>
```

For group *a*, the following messages are propagated to the Out terminal after 5 seconds:

```
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>6</seq></doc>
```

For group *b*, the following message is propagated to the Out terminal after 5 seconds:

```
<doc><grp>b<grp><seq>0</seq></doc>
```

No more messages are received before a missing message timeout occurs, at which point the following messages are propagated to the Expire terminal:

```
<doc><grp>b<grp><seq>2</seq></doc>
```

```
<doc><grp>b<grp><seq>3</seq></doc>
```

Using predicate set on the Resequence node

Select `Predicate` and specify an XPath expression to calculate whether the message is the first in the sequence. The predicate evaluates to either True or False, and messages continue to be collected while the expression evaluates to False. When the expression of a message is evaluated to True, it indicates that the message is the first in the sequence.

For example, you might specify the following XPath expression:

```
/Employee/EmpStartSeq="10"
```

When the input message field *EmpStartSeq* contains the value *10*, the start of sequence predicate is evaluated to True, and the message is identified as the first in the sequence:

```
<Employee>
 <EmpStartSeq>10</EmpStartSeq>
</Employee>
```

Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the following way:

Table 73.

Returned data type	**True**	**False**
Boolean	True	False
Numeric	Any non-zero value	0 or 0.0
String	Any string matching `true` (case-insensitive)	Any string not matching `true` (case insensitive)
NodeSet	Never	Always

When a message evaluates the expression to True (and is therefore identified as the start of the sequence), the node checks that the message has the smallest sequence number collected up to that point. If messages are found with lower sequence numbers, an exception is thrown.

When the first message that evaluates to true has been processed successfully, the XPath expressions of subsequent messages are not checked. If a message arrives with a lower sequence number than the message that was identified as the start of the sequence, an exception is thrown.

Using a Sequence node

When you use the Sequence node to add sequence numbers to messages in the message flow, you use the `Start of sequence definition` property to specify a literal number that is to be used as the starting sequence number. The value can be any positive or negative integer in the range -9223372036854775807 to 9223372036854775807.

The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node, starting with the sequence number that you define in the `Start of sequence definition` property. However, this value can be overridden by the value of the *StartOfSequenceNumber* field in the *LocalEnvironment* of the incoming message. For example: `InputLocalEnvironment.Sequence.StartOfSequenceNumber = 10`.

Ending a message sequence

The end of a message sequence is determined by the `End of sequence definition` property on the Resequence and Sequence nodes.

Using a Resequence node

When you use the Resequence node to reorder messages in a message flow, you use the `End of sequence definition` property on the Resequence node to define how the reordered message sequence will end. You can specify the end sequence number in the Resequence node in one of the following ways:

As a literal number

Select `Literal` to specify a literal sequence number as the end of the sequence. This value can be any positive or negative numeric value in the range -9223372036854775807 to 9223372036854775807. When a message with the specified sequence number arrives, the sequence group is closed. If there are any messages missing from the sequence, the sequence group remains open for the period of time specified by the `Missing message timeout` property.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`
- `Start of sequence definition` property set to `Literal` with a value of `0`.
- `End of sequence definition` property set to `Literal` with a value of 6. This value means that the group will be closed when a message with the sequence number 6 is received and when all earlier messages in the sequence have been received (or the missing message timeout expires).

The following messages are received by the Resequence node:

```
<doc><grp>a<grp><seq>6</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>0</seq></doc>
```

The following messages are propagated to the Out terminal:

```
<doc><grp>a<grp><seq>0</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>4</seq></doc>
<doc><grp>a<grp><seq>5</seq></doc>
<doc><grp>a<grp><seq>6</seq></doc>
```

When the sequence group has closed, any further messages arriving at the node for that sequence group are processed as part of a new instance of the group.

By an automatic timeout

Select `Automatic` and specify the length of time (in seconds) during which the node waits for new messages to arrive onto an empty queue. Each time the queue of messages waiting to be propagated is empty, the Resequence node starts a timer, which expires after the specified number of seconds. If no new messages arrive within the specified time, the sequence group is closed, and any new messages that arrive subsequently are treated as part of a new group. If new messages arrive at the node within the specified time limit, the timer is reset.

Using predicate set on the Resequence node

Select `Predicate` and specify an XPath expression to calculate whether the message is the last in the sequence. The predicate evaluates to either True or False, and messages continue to be collected while the expression evaluates to False. When the expression of a message is evaluated to True, it indicates that the message is the last in the sequence and the sequence group is closed. However,

if earlier messages are missing from the sequence group, the sequence group remains open for the period of time specified by the `Missing message timeout` property.

When the sequence group has closed, any further messages arriving at the node for that sequence group are processed as part of a new instance of the group.

Typically, the XPath expression evaluates to a Boolean; however, if other data types are returned, the predicate is determined in the following way:

Table 74.

Returned data type	**True**	**False**
Boolean	True	False
Numeric	Any non-zero value	0 or 0.0
String	Any string matching `true` (case-insensitive)	Any string not matching `true` (case insensitive)
NodeSet	Never	Always

When a message evaluates the expression to True (and is therefore identified as the last message in the sequence), the node checks that the message has the highest sequence number collected up to that point. If messages are found with higher sequence numbers, an exception is thrown.

Using a Sequence node

When you use the Sequence node to add sequence numbers to messages, you use the `End of sequence definition` property to define how the message sequence will end. The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node, ending with the sequence number that you define in the `End of sequence definition` property.

You can specify the end sequence number in the Sequence node in one of the following ways:

As a literal number
Select `Literal` and specify a positive or negative numeric value as the end of sequence number (for example, 15). The value must be in the range -9223372036854775807 to 9223372036854775807. The monotonically increasing sequence ends when it reaches the end of sequence number specified by this property.

Using predicate set on the Sequence node
Select `Predicate` to specify that the sequence ends when the XPath expression evaluates to True. For more information, see the table above.

By an automatic timeout
Select `Automatic` and specify the length of time (in seconds) during which the node waits for new messages after a message for the sequence group has been propagated. If a new message arrives, the timer is canceled and reset when the message is propagated. If no new messages arrive within the specified time, the sequence group is closed, and any new messages that arrive subsequently are treated as part of a new group.

Duplicate message processing

Each sequence number within a sequence group must be unique.

If a Resequence node receives a message containing a sequence number that has already been processed in the current sequence group, an exception is thrown (BIP4821) and the duplicate message is propagated to the Failure terminal. An exception occurs only if the duplicate message arrives after the first message with that sequence number has arrived and before the sequence group closes.

For example, assume that you have a Resequence node with the following properties:

- `Path to sequence number` property with a value of `/doc/seq`
- `Path to sequence group identifier` property with a value of `/doc/grp`

- `Start of sequence definition` property set to `Literal` with a value of `1`
- `End of sequence definition` property set to `Literal` with a value of `3`

The following messages arrive at the Resequence node:

```
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>2</seq>/doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
```

The following messages are propagated to the Out terminal:

```
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq></doc>
<doc><grp>a<grp><seq>3</seq></doc>
<doc><grp>a<grp><seq>1</seq></doc>
<doc><grp>a<grp><seq>2</seq>/doc>
<doc><grp>a<grp><seq>3</seq></doc>
```

The second occurrence of the input message with sequence number 2 causes an exception to be thrown and is propagated to the Failure terminal:

```
<doc><grp>a<grp><seq>2</seq>/doc>
```

Adding sequence numbers to messages

You can add sequence numbers to messages entering a message flow by using the Sequence node.

Before you begin

Read the concept topic about "Message sequencing" on page 1860.

About this task

The Sequence node allocates a monotonically increasing sequence number for each input message that arrives at the node. As each message arrives at the Sequence node, the sequence number is incremented and stored with the message in the location specified by the `Path to store sequence number` property. The allocation of sequence numbers continues until the sequence ends, as specified by the `End of sequence definition` property.

You can divide input messages into independent sequence groups, based on an identifier defined in the message. Each group has a separate group identifier, and the sequence of messages within each group is managed independently.

The Sequence node allocates a sequence number to each message in the sequence group, and the next sequence number in the group is not allocated until the current message in the group has finished processing (either by being committed or rolled back). This ensures that sequencing is maintained for the group when there are multiple threads in the message flow.

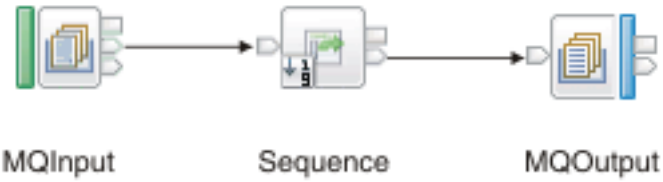
If you need to save the message with the newly assigned sequence number (for example, if you need to save the message to IBM MQ for processing by another flow), and if there is no convenient place in the message to save the sequence number, you can add an MQRFH2 header to the message before the Sequence node, and set the sequence number in a field in the `usr` folder.

Multiple sequence groups can be managed independently, in parallel, and sequence group state is preserved when the integration node is restarted.

The following steps show how to create a message flow that adds a sequence number to each message in a sequence group.

Procedure

1. Create a message flow containing an MQInput node, a Sequence node, and an MQOutput node.
2. Connect the Out terminal of the MQInput node to the In terminal of the Sequence node.
3. Connect the Out terminal of the Sequence node to the In terminal of the MQOutput node.



4. On the MQInput node, specify the source of input messages for the node by setting the `Queue name` property (on the **Basic** tab) to the name of an IBM MQ queue, from which the MQInput node retrieves messages. For example: `SEQ.TASK1.IN1`.
5. Set the following properties of the Sequence node:
   a) On the **Basic** tab, set the following properties:
   - Set the `Path to store sequence number` property to the location in the message where the sequence number is to be set. For example, `$OutputBody/doc/seq`. The sequence number is also set in the local environment, with the *LocalEnvironment.Sequence.Number* variable.
   - Set the `Path to sequence group identifier` property to the location of the sequence group identifier in the message. For example, `$InputBody/doc/grp`. The sequence group identifier is also copied to the local environment, with the *LocalEnvironment.Sequence.Group* variable.
   - Set the `Start of sequence definition` property to `Literal` with the required starting value; for example, `0`.

     Although the starting sequence number must be specified by a literal number, the value can be overridden in the local environment by the *LocalEnvironment.Sequence.StartOfSequenceNumber* variable.

     The start of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.Start* variable, which takes a Boolean value.
   - Set the `End of sequence definition` property to one of the following values:
     - `Automatic` with the required timeout value; for example, `60`. This value specifies that the sequence group is closed automatically when the message queue in the node has been empty for 60 seconds.
     - `Literal` with the required end value; for example, `100`. This value specifies that the sequence group is closed when the message with the sequence number `100` is processed.
     - `Predicate` with the required XPATH expression; for example, `$InputBody/doc/endFlag`. This value specifies that the sequence group is closed when the `$InputBody/doc/endFlag` predicate evaluates to True (`$InputBody/doc/endFlag=True`).

     The end of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.End* variable, which takes a Boolean value.

   b) On the **Advanced** tab, set the `Persistence mode` property to `Non-persistent`.

   Select `Persistent` if you want the sequence to be preserved if the queue manager is restarted.
6. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue to which the MQOutput node sends messages. For example: `SEQ.TASK1.OUT1`.
7. Save your message flow.

Reordering messages in a message flow

When messages entering a message flow contain sequence numbers and a group ID, you can use the Resequence node to re-establish the sequential order of the messages before propagating them through the message flow.

Before you begin

Read the concept topic about "Message sequencing" on page 1860.

Ensure that each message contains a monotonically increasing sequence number.

About this task

The sequence number might have been added to the message by the Sequence node or it might be another integer field in the message.

When the Resequence node receives an input message, it propagates the message only if it is the next one in the sequence. If the message is not next in the sequence, the Resequence node stores it until further messages arrive that allow the node to correct the sequence, at which point the node propagates the stored message. If a message fails to arrive, preventing the Resequence node from completing the sequence, the remaining messages are processed according to the way in which the Resequence node has been configured. For more information about this configuration, see "Handling missing messages" on page 1873.

A transaction break occurs at the Resequence node. When a message is delivered to the Resequence node, control is returned to the previous node in the message flow. All messages that are propagated from the Resequence node are propagated in a new transaction, even if the sequence is complete. For more information, see node.

The message sequence is preserved when the integration node is restarted. If the `Persistent` option is selected on the **Advanced** tab of the Resequence node, the sequence is also preserved when the queue manager is restarted.

The following steps show how to create a message flow that enables you to re-establish the sequential order of the messages in a sequence group:

Procedure

1. Switch to the Integration Development perspective.
2. Create a message flow containing an MQInput node, a Resequence node, and an MQOutput node.
3. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
4. Connect the Out terminal of the Resequence node to the In terminal of the MQOutput node.

5. On the MQInput node, specify the source of input messages for the node by setting the `Queue name` property (on the **Basic** tab) to the name of an IBM MQ queue, from which the MQInput node retrieves messages. For example: `RESEQ.TASK1.IN1`.

6. Set the following properties of the Resequence node:
   a) On the **Basic** tab, set the following properties:
      - Set the `Path to sequence number` property to the location of the sequence number in the message. For example, `$InputBody/doc/seq`. The sequence number is also set in the local environment, with the *LocalEnvironment.Sequence.Number* variable.
      - Set the `Path to sequence group identifier` property to the location of the sequence group identifier in the message. For example, `$InputBody/doc/grp`. The sequence group identifier is also copied to the local environment, with the *LocalEnvironment.Sequence.Group* variable.
      - Set the `Start of sequence definition` property to the first sequence number in the group. For example, select `Literal` with a value of `0`.
      - Set the `End of sequence definition` property to `Automatic` with the required timeout value; for example, `60`. This value specifies that the sequence group is closed automatically when the message queue in the node has been empty for 60 seconds. The end of sequence message is indicated in the local environment by the *LocalEnvironment.Sequence.End* variable, which takes a Boolean value.
      - Set the `Missing message timeout` property to `10`. This value specifies that the Resequence node will wait for a missing message for 10 seconds before propagating subsequent messages in the sequence group to the Expire terminal. When the subsequent message is propagated, the sequence numbers of any missing (timed-out) messages are copied to the local environment, as *LocalEnvironment.Sequence.Missing* variables.
7. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to the name of a WebSphere MQ queue to which the MQOutput node sends messages. For example: `SEQ.TASK1.OUT1`.
8. Save your message flow.

Maintaining the sequential order of messages

You can maintain the order in which messages enter a message flow by using the Sequence and Resequence nodes.

Before you begin

Read the concept topic about "Message sequencing" on page 1860.

About this task

When messages have been processed in a message flow, they might be propagated in a different order to the order in which they arrived at the input node, so the original message sequence can be altered. You can use the Sequence and Resequence nodes to rearrange the messages into sequential order based on a sequence number in the message, restoring the original sequence in which they entered the input node.

If the messages contain sequence numbers, you can use the Resequence node to re-establish the order in which they arrived at the input node. If the messages do not contain sequence numbers, you can use a Sequence node to apply sequence numbers to the messages, before reordering them into sequential order using the Resequence node.

Procedure

- If the messages contain sequence numbers, rearrange the messages into sequential order by creating a message flow containing a Resequence node. See "Reordering messages in a message flow" on page 1868.

- If the messages do not contain sequence numbers, establish the sequential order of the messages by completing the following steps:
  a) Create a message flow containing a Sequence node, which you can use to apply sequence numbers to the messages. See “Adding sequence numbers to messages” on page 1866.
  b) Create a message flow containing a Resequence node, which you can use to re-establish the sequential order of the messages. See “Reordering messages in a message flow” on page 1868.

Message sequencing scenario 1

Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

Before you begin

Read the concept topic about “Message sequencing” on page 1860.

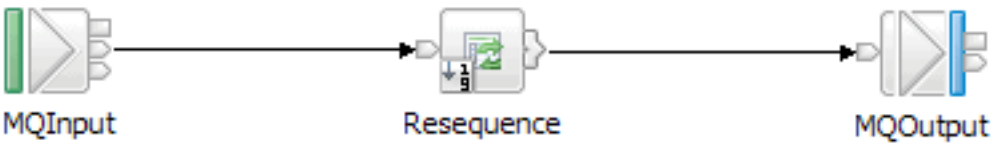
About this task

In this example task, the sequence is defined as starting at sequence number 0 (which is the default) and finishing at sequence number 8. The message flow is configured so that messages can arrive out of order but cannot be propagated out of order, even if one of the sequence numbers never arrives.

The following steps show how to write a message flow that can receive the XML document from WebSphere MQ, reorder the messages based on a sequence number in an XML message (in this example the path `$Root/XMLNSC/Doc/SeqNo` is used) and write it to a IBM MQ queue:

Procedure

1. Create a message flow called Resequence_Task1, containing an MQInput node, a Resequence node, and an MQOutput node.
   For more information about how to do this, see “Creating a message flow” on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
3. Connect the Out terminal of the Resequence node to the In terminal of the MQOutput node.



4. Set the following properties of the MQInput node:
   a) On the **Basic** tab, set the `Queue name` property to `RESEQUENCE_TASK1_IN1`
   b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
5. On the Resequence node, set the following properties on the **Basic** tab:
   a) Set the `Path to sequence number` property to `$Root/XMLNSC/Doc/SeqNo`
   b) Set the `End of sequence definition` property to `Literal` with a value of 8.
6. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `RESEQUENCE_TASK1_OUT1`.
7. Save the message flow.

Results

Message processing in the message flow:

This section describes the way in which the Resequence node processes the messages that enter the message flow.

1. The following messages arrive on the WebSphere MQ queue RESEQUENCE_TASK1_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>5</SeqNo></Doc>,
<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>8</SeqNo></Doc>,
```

2. The Resequence node first receives the message with sequence number 0. The Resequence node creates a new sequence group to manage the reordering process; the new sequence group is a default group because no sequence group has been defined in the message. The message (with sequence number 0) is the first one in the sequence, so it is propagated to the Out terminal.
3. The Resequence node then receives the rest of the messages up to and including sequence number 8, and propagates them in the order in which they arrived. Each message is stored before it is propagated, and a different transaction is used for propagating the messages downstream from the Resequence node.
4. When the message containing sequence number 8 is processed, the sequence group is closed. Any new message in the same group that arrives later causes a new group to be created.
5. The next messages arrive on the RESEQUENCE_TASK1_IN1 queue:

```
<Doc><SeqNo>8</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>6</SeqNo></Doc>,
<Doc><SeqNo>5</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>3</SeqNo></Doc>,
<Doc><SeqNo>2</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>0</SeqNo></Doc>,
```

6. The Resequence node first receives the message containing sequence number 8. At this point, the Resequence node creates a new (default) sequence group. The message is the last one in the sequence so it is not propagated but is stored internally.
7. The Resequence node then receives the rest of the messages up to and including sequence number 0. All the messages are stored and none are propagated until the first number in the sequence is received (in this case, sequence number 0).
8. When the message with sequence number 0 is received, it is propagated down the message flow, followed by each of the other messages, in order, until the final message (sequence number 8) is propagated. At this point, the sequence group is closed again.

Message sequencing scenario 2

Change the sequence of messages received from WebSphere MQ, so that they are propagated in the correct order using a sequence number defined in an XML message.

Before you begin

Read the concept topic about “Message sequencing” on page 1860.

About this task

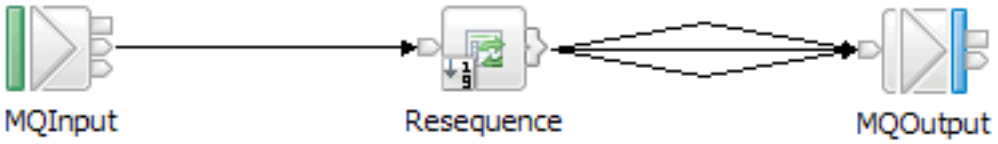
In this example task, the sequence is defined as starting at sequence number 0 (which is the default) and finishing when it has received no new messages in the group for 60 seconds. The message flow is configured so that messages can arrive out of order but the Resequence node attempts to propagate them in the correct order. If a required message in the sequence fails to arrive for 10 seconds, the Resequence node skips the missing message and propagates the next message in the sequence, even though it is now out of order.

The following steps show how to write a message flow that can receive the XML document from WebSphere MQ, reorder the messages based on a sequence number in an XML message (in this example the path `$Root/XMLNSC/Doc/SeqNo` is used) and write it to a IBM MQ queue:

Procedure

1. Create a message flow called Resequence_Task2, containing an MQInput node, a Resequence node, and an MQOutput node.

   For more information about how to do this, see "Creating a message flow" on page 574.
2. Connect the Out terminal of the MQInput node to the In terminal of the Resequence node.
3. Connect the Out, Missing, and Expire terminals of the Resequence node to the In terminal of the MQOutput node.



4. Set the following properties of the MQInput node:
   a) On the **Basic** tab, set the `Queue name` property to `RESEQUENCE_TASK2_IN1`
   b) On the **Input Message Parsing** tab, set the `Message domain` property to `XMLNSC`.
5. On the Resequence node, set the following properties on the **Basic** tab:
   a) Set the `Path to sequence number` property to `$Root/XMLNSC/Doc/SeqNo`
   b) Set the `Missing message timeout` property to `10`
   c) Set the `End of sequence definition` property to `Automatic` with a value of `60`.
6. On the MQOutput node, set the `Queue name` property (on the **Basic** tab) to `RESEQUENCE_TASK2_OUT1`.
7. Save the message flow.

Results

Message processing in the message flow:

This section describes the way in which the Resequence node processes the messages that enter the message flow:

1. The following messages arrive on the WebSphere MQ queue RESEQUENCE_TASK2_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>1</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>5</SeqNo></Doc>,
<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,<Doc><SeqNo>8</SeqNo></Doc>,
```

2. The Resequence node first receives the message with sequence number 0. The Resequence node creates a new sequence group to manage the reordering process; the new sequence group is a default group because no sequence group has been defined in the message. The message (with sequence number 0) is the first one in the sequence, so it is propagated to the Out terminal.
3. The Resequence node then receives the rest of the messages up to and including sequence number 8, and propagates them in the order in which they arrived. Each message is stored before it is propagated, and a different transaction is used for propagating the messages downstream from the Resequence node.
4. Sixty seconds elapse and the sequence group is closed.
5. The next messages arrive on the WebSphere MQ queue RESEQUENCE_TASK2_IN1:

```
<Doc><SeqNo>0</SeqNo></Doc>,<Doc><SeqNo>3</SeqNo></Doc>,<Doc><SeqNo>2</SeqNo></Doc>,
<Doc><SeqNo>4</SeqNo></Doc>,<Doc><SeqNo>6</SeqNo></Doc>,<Doc><SeqNo>7</SeqNo></Doc>,
```

6. The Resequence node first receives the message containing sequence number 0. At this point, the Resequence node creates a new (default) sequence group and propagates the message with sequence number 0 to the Out terminal.
7. The next message contains sequence number 3, which is out of sequence. The `Missing message timeout` property is set on the Resequence node with a value of 10 seconds, and the timer starts.
8. Sequence number 2 arrives, which is still not the next required message but it is lower than the lowest message currently stored, so the missing message timer is reset.
9. The Resequence node then receives the rest of the messages, all of which have a higher sequence number than 2, so they are stored but not propagated. The missing message timer is not reset this time because each number is later in the sequence than number 2.
10. Sequence number 1 fails to arrive during the specified period, and after 10 seconds the timeout period expires.
11. The Resequence node propagates all the messages that are stored, starting with the lowest sequence number (in this case, 2) followed by all other messages up to the next missing number in the sequence (in this case, messages with sequence numbers 3 and 4). Messages 2, 3, and 4 are propagated to the Expire terminal.
12. The Resequence node is moved to an unordered state for this sequence group, and will not propagate any messages in this group to the Out terminal. The Resequence node remains in the unordered state for this sequence group until the group expires and is closed.
13. When another 10 seconds has passed, the missing message timer expires again and messages 6 and 7 are propagated to the Expire terminal. At this point there are no more missing messages so the missing message timer is not restarted.
14. The group expiry timer is started and the group is closed after 60 seconds have passed. The group expiry timer is never started when a missing message timer is running.

Handling missing messages

You can configure the Resequence node to control how missing messages in a sequence are processed.

About this task

You can configure the Resequence node to time out if a message in the sequence fails to arrive in a specified period of time, and you can specify how subsequent messages are processed if a message is missing.

You can use the `Missing message timeout` property of the Resequence node to specify how long (in seconds) the node waits for the next message in the sequence before it moves on to the next message. Messages that arrive within the specified time limit are propagated in sequential order to the Out terminal. When the specified time limit has been exceeded, the messages are propagated in sequential order to the Expire terminal. Subsequent messages in the sequence group are also routed to the Expire terminal. If the missing message eventually arrives, it is propagated to the Missing terminal.

You can configure the Resequence node for any of the following scenarios:

- **The message sequence must always be maintained**

  If a message goes missing, you can route all subsequent messages to a holding queue after a specified timeout period. To configure the Resequence node in this way, wire the Out terminal to the main-line flow and wire the Expire and Missing terminals to separate branches for re-queueing.

- **Missing messages are allowed in the message sequence but all other messages must remain in sequence**

  If a message goes missing, you can pass over it and continue processing the rest of the sequence. If the missing message eventually arrives, you can either discard it or process it separately from the main-line processing. To configure the Resequence node in this way, wire the Out and Expire terminals to the main-line flow and leave the Missing terminal unwired (to discard the message) or wire it to a separate branch of the message flow.

- **Occasional out-of-sequence messages are allowed**

  You want the message flow to process the messages in sequential order, but you are prepared to tolerate occasional messages out of order. To configure the Resequence node in this way, wire the Out, Expire, and Missing terminals to the main-line message flow.

Configuring the storage of events for Resequence nodes

You can use a Resequence policy to control the storage of events for Resequence nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using Resequence nodes on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the Resequence node. If you are using Resequence nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing Resequence nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

By default, the storage queues used by all Resequence nodes are:

- SYSTEM.BROKER.EDA.EVENTS
- SYSTEM.BROKER.EDA.COLLECTIONS

These queues are also used by the Collector node.

However, you can control the queues that are used by different Resequence nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Resequence policy to specify the names of those queues for storing events.

Follow these steps to specify the queues that are used to store event states, and to set the timeout and the start and end of the sequence:

Procedure

1. Create the storage queues to be used by the Resequence node.

   The following queues are required:

   - SYSTEM.BROKER.EDA.*QueuePrefix*.EVENTS
   - SYSTEM.BROKER.EDA.*QueuePrefix*.COLLECTIONS

   The *QueuePrefix* variable can contain any characters that are valid in a IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET.1 are valid queue prefixes, but .SET1 and SET1. are invalid.

   If you do not create the storage queues, IBM App Connect Enterprise creates the set of queues when the node is deployed; these queues are based on the default queues. If the queues cannot be created, the message flow is not deployed.

2. Create a Resequence policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either a specific sequence or with all sequences in an integration server. If you are creating a policy to be used with a specific sequence, ensure that the name of the policy is the same as the name that you specify in the `Policy` property on the Resequence node.

      To specify a default Resequence policy for all message flows that are deployed to an integration server, set the **Resequence** property in the `server.conf.yaml` file to the name of a Resequence policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*
   b) Set the **Queue prefix** property of the Resequence policy to the required value (see Resequence policy).
   c) Optional: Set the **Missing message timeout**, **Start of sequence**, and **End of sequence** properties of the Resequence policy.

   If you delete the Resequence policy, the storage queues are not deleted automatically when the policy is deleted, so you must delete them separately.
3. In the Resequence node, if the policy is to be used for a specific sequence, specify the name of the policy on the **Advanced** tab; for example, `myResequenceService`.

   If you do not set the `Policy` property, and if there is a default Resequence policy specified in the `server.conf.yaml` file, that policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Configuring timeout flows

Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. If you are using timeout flows on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by the TimeoutControl and TimeoutNotification nodes. If you are using these nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see "Using a remote default queue manager" on page 750 and "Configuring an integration server to use a remote default queue manager" on page 751.

The system queues that hold the state information are owned by the queue manager that is specified on the integration server. You associate a queue manager with the integration server by specifying the queue manager name on the **defaultQueueManager** property in the `server.conf.yaml` configuration file; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

You must create the system queues by running a script command, as described in "Creating the default system queues on an IBM MQ queue manager" on page 99.

For more information about using IBM MQ with IBM App Connect Enterprise, see "Enhanced flexibility in interactions with IBM MQ " on page 26 and "Installing IBM MQ" on page 96.

The following topics show how the timeout nodes can be used in a message flow:

- “Sending timeout request messages” on page 1876
- “Sending a message after a timed interval” on page 1878
- “Sending a message multiple times after a specified start time” on page 1879
- “Automatically generating messages to drive a flow” on page 1880
- “Configuring the storage of events for timeout nodes” on page 234

Sending timeout request messages

To set a controlled timeout, send a message with a set of elements with well known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

Elements and format

The following example shows the elements and format of a timeout request message, showing the well known names and permissible values.

```
<TimeoutRequest>
  <Action>SET | CANCEL</Action>
  <Identifier>String (any alphanumeric string)</Identifier>
  <StartDate>String (TODAY | yyyy-mm-dd)</StartDate>
  <StartTime>String (NOW | hh:mm:ss)</StartTime>
  <Interval>Integer (seconds)</Interval>
  <Count>Integer (greater than 0 or -1)</Count>
  <IgnoreMissed>TRUE | FALSE</IgnoreMissed>
  <AllowOverwrite>TRUE | FALSE</AllowOverwrite>
</TimeoutRequest>
```

Message fields

The following table describes the fields in the message. The column headed M indicates whether the property is mandatory, and the column headed C indicates whether the property is configurable.

Property	M	C	Default	Description
Action	Yes	No	None	Set this element to either SET or CANCEL. An error is generated if you omit this element or set it to a different value. If you set it to CANCEL, the only other element that is required is the Identifier, which must match the Identifier of the TimeoutRequest that is to be canceled.
Identifier	Yes	No	None	Enter an alphanumeric string. An error is generated if you omit this element.
StartDate	No	No	TODAY	Set this element to TODAY or to a date specified in yyyy-mm-dd format. The default value is TODAY.
StartTime	No	No	NOW	Set this element to NOW or to a time in the future specified in hh:mm:ss format. The default value is NOW. StartTime is assumed to be the integration node's local time. The start time can be calculated by adding an interval to the current time. If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is now or a time in the future.
Interval	No	Yes	0	Set this element to an integer that specifies the number of seconds between propagations of the message. The default value is 0.

Property	M	C	Default	Description
Count	No	Yes	1	Set this element to an integer value that is either greater than 0 or is -1 (which specifies a timeout request that never expires). The default value is 1.
IgnoreMissed	Yes	No	TRUE	Set this element to TRUE or FALSE to control whether timeouts that occur while either the integration node or the timeout notification flow is stopped, are processed the next time that the integration node or timeout notification flow is started. The default value is TRUE, which means that missed timeouts are ignored by the TimeoutNotification node when the integration node or message flow is started. If this value is set to FALSE, the missed timeouts are all processed immediately by the TimeoutNotification node when the flow is started. You must set the `Request Persistence` property of the TimeoutControl node to Yes or Automatic (with the originating request message being persistent) for the stored timeouts to persist beyond the restart of the integration node or the timeout notification flow.
AllowOverwrite	N	N	TRUE	Set this element to TRUE or FALSE, to specify whether subsequent timeout requests with a matching `Identifier` can overwrite this timeout request. The default value is TRUE.

How the TimeoutControl node uses these values

Set the `Request Location` on the TimeoutControl node to `InputRoot.XML.TimeoutRequest` to read these properties. If you want to obtain properties from a different part of your message, specify the appropriate correlation name for the parent element for the properties. The correlation name for the parent element can be in the local environment.

For details of how the TimeoutControl uses these values, see node

Working with the predefined schema definition of an XML timeout request message
A predefined schema definition of an XML timeout request message is provided in the IBM App Connect Enterprise Toolkit.

About this task

Take the following steps to review the definition or to define it in a message set.

1. Create or select a message set project that contains the message set.
2. Create a new message definition file (**File** > **New** > **Message Definition File From...**).
3. Select **IBM supplied message** and click **Next**.
4. Expand the tree for **Message Brokers IBM supplied Message Definitions**.
5. Select **Message Broker Timeout Request** and click **Finish**.

Example XML timeout request message
The format used here is XML, but you can use any format that is supported by an installed parser.

```
<TimeoutRequest>
  <Action>SET</Action>
  <Identifier>TenTimes</Identifier>
  <StartDate>TODAY</StartDate>
  <StartTime>NOW</StartTime>
  <Interval>10</Interval>
  <Count>10</Count>
  <IgnoreMissed>TRUE</IgnoreMissed>
  <AllowOverwrite>TRUE</AllowOverwrite>
</TimeoutRequest>
```

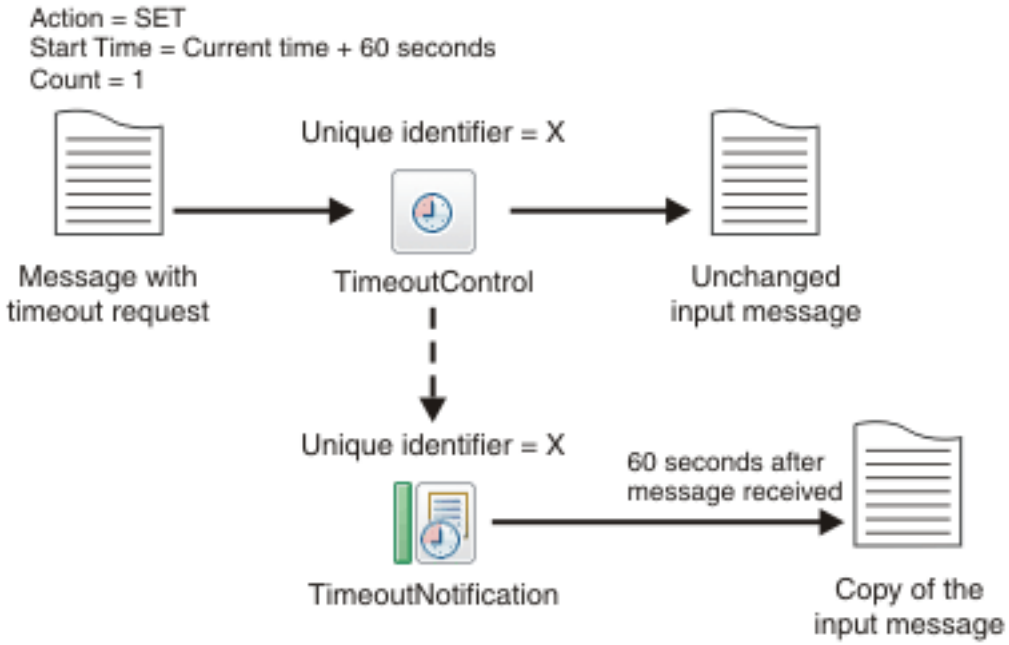
Sending a message after a timed interval

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow after a timed interval.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow 60 seconds after the message is received.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that the TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *current time + 60*
Count set to *1*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same `Unique identifier` as the TimeoutControl node propagates a copy of the message to the message flow 60 seconds after the message was received.

Timeout request messages are stored for processing on a queue used by the TimeoutNotification node. By default this queue is the `SYSTEM.BROKER.TIMEOUT.QUEUE`. However, you can use a Timer policy to specify an alternative timeout queue, which provides greater control over the storage of messages. For information about using an alternative timeout queue, see “Configuring the storage of events for timeout nodes” on page 234.

If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node

rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate. Therefore, ensure that the start time in the timeout request message is a time in the future.

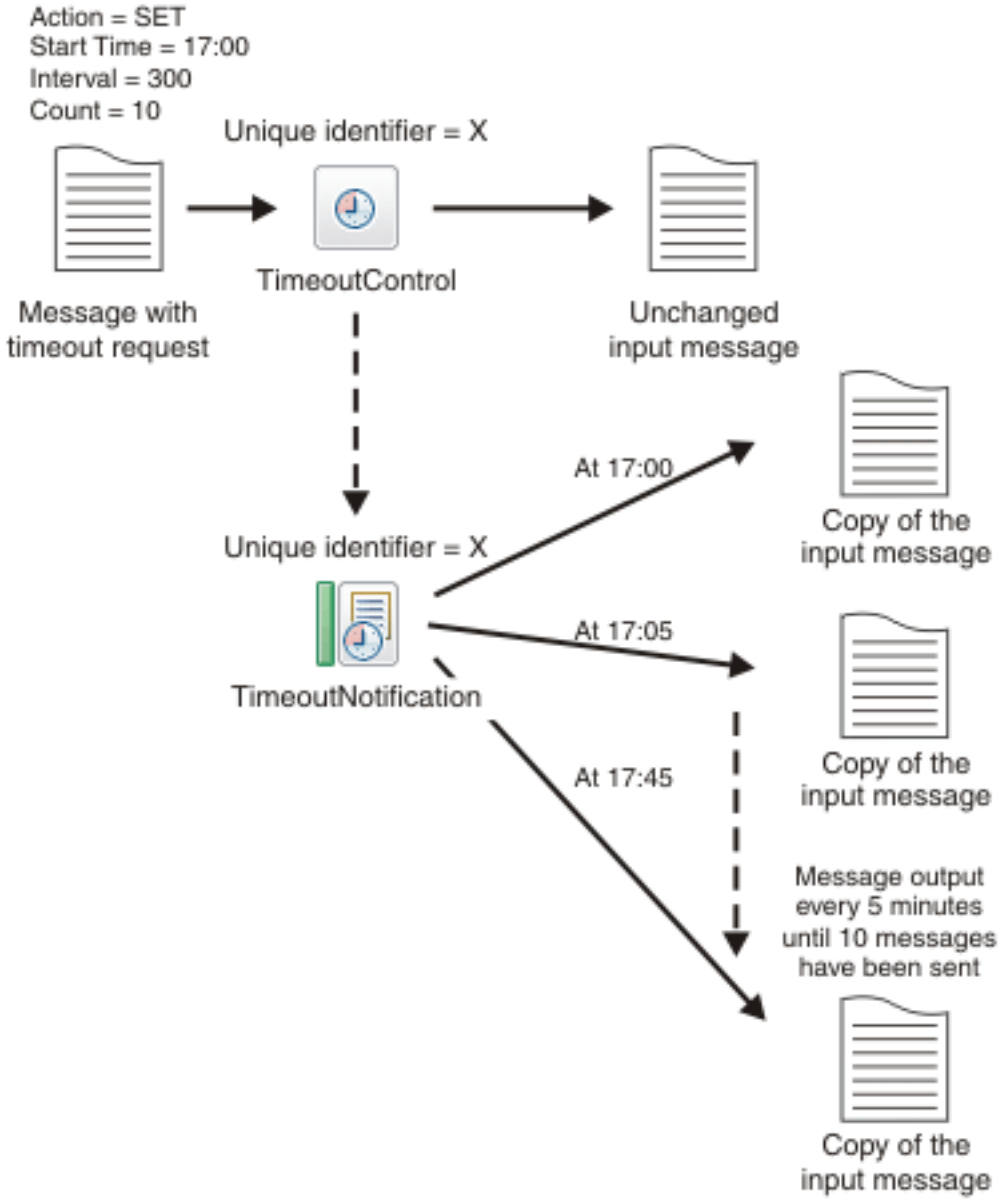
Sending a message multiple times after a specified start time

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow multiple times after a specified start time.

Aim

Use TimeoutControl and TimeoutNotification nodes to send a message into a message flow at 17:00 hours, then send the message again every 5 minutes until the message has been sent 10 times.

Description of the flow



The diagram shows the path of a message that contains a timeout request through a TimeoutControl node. A TimeoutNotification node with an identifier matching the TimeoutControl node then processes the timeout request. The diagram also shows the message that he TimeoutNotification node produces after processing the timeout request.

The message comes into the TimeoutControl node with the following values set in the timeout request section of the message:

Action set to *SET*
Start Time set to *17:00*
Interval set to *300*
Count set to *10*

The TimeoutControl node validates the timeout request; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow. If the request is valid, the TimeoutNotification node with the same `Unique identifier` as the TimeoutControl node propagates a copy of the message to the message flow at 17:00. The message is sent again after an interval of 300 seconds, at 17:05. and every 300 seconds until the message has been sent 10 times, as the *Count* value in the timeout request specifies.

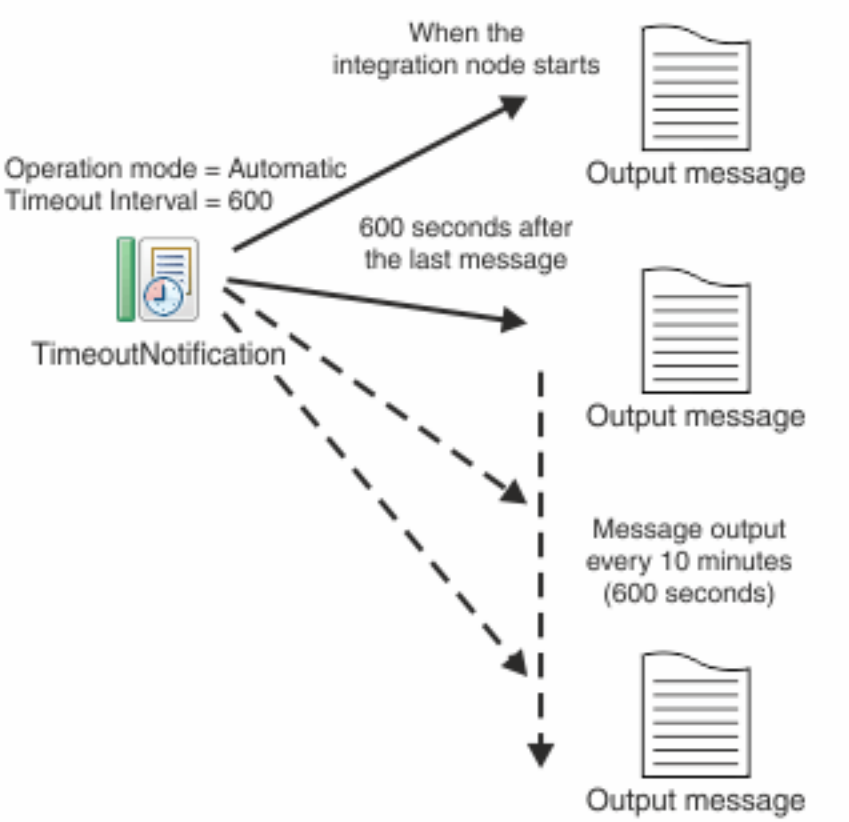
Automatically generating messages to drive a flow

Using the TimeoutNotification node to automatically send a message into a message flow.

Aim

Use the TimeoutNotification node to automatically send a message into a message flow every 10 minutes.

Description of the flow



The diagram shows a TimeoutNotification node automatically generating messages and propagating them every 10 minutes. To get the TimeoutNotification node to automatically generate messages, set the `Operation Mode` property of the node to automatic and specify a value for the `Timeout Interval` property. In this example the TimeoutNotification node has the following properties:

`Operation Mode` set to automatic
`Timeout Interval` set to 600

You can also use a Timer policy to control the timeout interval. A value that is set by the **Timeout interval** property of the Timer policy overrides the value that is specified on the TimeoutNotification node (see Timer policy).

When the integration node has started, the TimeoutNotification node sends a message every 10 minutes (600 seconds). This message contains only a properties folder and a LocalEnvironment folder. A Compute node can then process this message to create a more meaningful message.

If the downstream processing is intensive, and the flow is still busy when the next timeout occurs, the TimeoutNotification node does not send a message immediately. When the flow completes current processing, all timeouts that were missed while the flow was busy are processed individually.

Configuring the storage of events for timeout nodes

You can use a Timer policy to control the storage of events for TimeoutNotification and TimeoutControl nodes.

About this task

Information about the state of in-flight messages is held on storage queues that are controlled by IBM MQ. The storage queues that hold the state information are owned by the queue manager that is associated with the integration server.

If you are using TimeoutControl and TimeoutNotification nodes on an integration server that is managed by an integration node, you must install IBM MQ on the same computer as your integration node in order to use the capabilities that are provided by these nodes. If you are using TimeoutControl and TimeoutNotification nodes on an independent integration server, you can use a remote default queue manager to control the system queues, without the need to install IBM MQ on the same machine as the integration server. Interactions between an independent integration server and IBM MQ can use a client connection to a remote queue manager, by using a default policy setting. For more information about using a remote default queue manager, see “Using a remote default queue manager” on page 750 and “Configuring an integration server to use a remote default queue manager” on page 751.

If the integration server has the necessary permissions to create the default system queues, they are created automatically when a flow containing TimeoutControl or TimeoutNotification nodes is deployed. If the default queues are not created automatically, you can create them manually by running the **iib_createqueues** command, as described in “Creating the default system queues on an IBM MQ queue manager” on page 99.

By default, the storage queue used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE. However, you can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer policy to specify the names of those queues for storing events.

Follow these steps to specify the queue that is used to store event states:

Procedure

1. Create the storage queue to be used by the timeout nodes.

   The following queue is required:

   - SYSTEM.BROKER.TIMEOUT.*QueuePrefix*.QUEUE

   The *QueuePrefix* variable can contain any characters that are valid in an IBM MQ queue name, but must be no longer than eight characters and must not begin or end with a period (.). For example, SET1 and SET`.`1 are valid queue prefixes, but `.`SET1 and SET1`.` are invalid.

   If you do not create the storage queue, IBM App Connect Enterprise creates the queue when the node is deployed; this queue is based on the default queue. If the queue cannot be created, the message flow is not deployed.

2. Create a Timer policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
   a) You can create a policy to be used with either specific timeout requests or with all timeout requests in an integration server. If the policy is to be used with specific timeout requests, create the policy with the same name as the `Unique identifier` property on the TimeoutNotification and TimeoutControl nodes.

      To specify a default Timer policy for all message flows that are deployed to an integration server, set the **`Timer`** property in the `server.conf.yaml` file to the name of a Timer policy. For information about setting properties in the `server.conf.yaml` file, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. If the default policy is in the default policy project, you do not need to specify the name of the policy project. If the default policy is in a non-default policy project, qualify the name of the policy with the name of the policy project in the format *{policyProjectName}:PolicyName*.
   b) Set the **`Queue prefix`** property of the Timer policy to the required value (see Timer policy).

   If you delete the Timer policy, the storage queue is not deleted automatically when the policy is deleted, so you must delete it separately.
3. In the TimeoutNotification and TimeoutControl nodes, ensure that the name of the Timer policy is the same as the name specified in the `Unique Identifier` property on the **Basic** tab; for example, `myTimer`. Specify the name of the policy project and the policy on the message flow node in the format *{policyProjectName}:PolicyName*. If there is no Timer policy with the same name as the `Unique Identifier`, and if there is a default Timer policy specified in the `server.conf.yaml` file, that Timer policy is used instead.

What to do next

The properties for the policy are not used by the integration server until you restart or redeploy the message flow, or restart the integration server.

Considering performance for timeout flows

When you design timeout flows, the decisions that you make can affect the performance of your integration nodes and applications.

You can use the timeout nodes TimeoutControl and TimeoutNotification in your message flows to control the way in which your message flows operate:

- Set the Operation Mode property of the TimeoutNotification node to Automatic. This setting causes a flow to be invoked at the interval that you specify in the Timeout Value property. If the downstream processing is intensive, and the flow is still busy when the next timeout occurs, the TimeoutNotification node does not send a message immediately. When the flow completes current processing, all timeouts that were missed while the flow was busy are processed individually.

  The value of the Additional Instances property of the message flow is ignored downstream of a TimeoutNotification node, and you cannot use this property to change the behavior of the flow.
- Use two associated flows to perform user-defined timeout processing. Set a timeout with a TimeoutControl node, and notify the flow using a TimeoutNotification node (which behaves like an input node to start a new message flow thread). If the downstream processing from the TimeoutNotification node is significant, requests that are set up in the TimeoutControl node can build up. These requests are subsequently processed individually by the TimeoutNotification node in a single thread.

  You cannot increase the Additional Instances property of the message flow if it starts with a TimeoutNotification node, therefore you cannot apply more threads to increase the capacity of the flow.

Although you can use a TimeoutNotification node to cause nodes in a message flow to poll for the next item of work, this approach forces a delay between each transaction, and typically does not provide an efficient solution. If you want to periodically check a resource for the next piece of work, and process it immediately, consider one or more of the following alternative solutions:

- Use a built-in input node.
- Write your own input node by using the user-defined node API (in Java or C).

- Consider purchasing an IBM or vendor-provided adapter which polls the subsystem you want, and triggers the flow.

A message flow that uses these options can process more work overall than it can if you implement a timeout solution, and incurs lower CPU cost, although your initial development costs might be slightly higher.

Handling errors in message flows

The integration node provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows by using message flow node terminals to provide your own error handling.

About this task

For example, you might design a message flow that expects certain errors that you want to process in a particular way. Another example is that your message flow updates a database, and must roll back those updates if other message processing does not complete successfully.

The options that you can use to do this are quite complex in some cases. The options that are provided for MQInput and TimeoutNotification nodes are extensive because these nodes deal with persistent messages and transactions. The MQInput node is also affected by configuration options for IBM MQ.

Because you can decide to handle different errors in different ways, there are no fixed procedures to describe. This section provides information about the principles of error handling, and the options that are available, and you must decide what combination of choices that you require in each situation based on the details that are provided in this section.

There are two general approaches to handling errors in a message flow:

- Failure checking

  Wire the Failure terminal of a node to explicitly check for any errors that occur within that node. If errors occur, an exception list is propagated to the Failure terminal. The inflight message remains the same as it was before the node was invoked.

  You can introduce more specialized error checking in nodes that can be customized by using ESQL. For example, you can create exit handlers within these nodes. For more information about using ESQL to create exit handlers, see DECLARE HANDLER statement.

- Catching exceptions

  If you do not wire a Failure terminal, a failure in the node is converted into an exception which is thrown from the node. Any changes that were made to the inflight message before the exception was thrown are reversed. The exception might cause the current transaction to be rolled back, which means that any updates to transactional resources are reversed. Root and Local Environment trees are rolled back in parallel. The Environment tree is not rolled back unless the entire transaction for the flow rolls back.

  You can prevent the transaction from being rolled back, and control the extent to which message changes are reversed, by including a TryCatch node in your message flow. If an exception is thrown beyond the Try terminal of the TryCatch node, then an exception list is propagated to the node's Catch terminal. The inflight message reverts to the state it was in before it reached the TryCatch node.

  Most message flow nodes have a Catch terminal. These nodes are typically at the start of a transaction, where an uncaught exception would cause a rollback. In these nodes, the Catch terminal behaves as though a TryCatch node was wired directly to the Out terminal. Use the Catch terminal to handle any exceptions that are thrown beyond the message flow node. Wire the Failure terminal to handle errors within the node itself.

You can choose one or more of these options in your message flows:

- Connect the Failure terminal of the node to a sequence of nodes that processes the node's internal exception (the fail flow).

- Connect the Catch terminal of the node to a sequence of nodes that processes exceptions that are generated beyond it (the catch flow).
- Insert one or more TryCatch nodes at specific points in the message flow to catch and process exceptions that are generated by the flow connected to the Try terminal.
- Include a Throw node, or code an ESQL THROW statement, to generate an exception.
- Ensure that all the messages received by an MQInput node are processed in a transaction, or are not processed in a transaction.
- Ensure that all the messages received by an MQInput node are persistent, or are not persistent.

If you include user-defined nodes in your message flow, you must see the information provided with the node to understand how you might handle errors with these nodes. The descriptions in this section cover only the built-in nodes.

When you design your error handling approach, consider the following factors:

- Most of the built-in nodes have Failure terminals. The exceptions are the AggregateControl, AggregateRequest, Input, Label, Output, Passthrough, Publication, Trace, and TryCatch nodes.

  When an exception is detected in a node, the message and the exception information are propagated to the node's Failure terminal. If the node does not have a Failure terminal, or it is not connected, the integration node throws an exception and returns control to the closest upstream node that can process it. This node might be a TryCatch node, an AggregateReply node, or the input node.

  If an MQInput node detects an internal error, its behavior is slightly different. If the Failure terminal is not connected, the MQInput node attempts to put the message to the input queue's backout queue, or if that is not defined, to the dead letter queue of the queue manager that is associated with the integration node.

  For more information, see "Handling MQInput node errors" on page 1886.
- Many built-in nodes have Catch terminals. These nodes are typically at the start of a transaction. For example:
  - Input nodes: FileInput, HTTPInput, JMSInput, MQInput, PeopleSoftInput, SAPInput, SiebelInput, SOAPInput, TCPIPClientInput, TCPIPServerInput
  - Output and response nodes: SOAPAsyncResponse
  - Routing nodes: AggregateReply, Collector, Resequence
  - Construction nodes: TryCatch
  - Timer nodes: TimeoutNotification

  A message is propagated to a Catch terminal only if it has first been propagated beyond the node (for example, to the nodes connected to the Out terminal).
- When a message is propagated to the Failure or Catch terminal, the node creates and populates a new exception list tree with an exception that represents the error that has occurred. The exception list is propagated as part of the message tree. When a message is propagated to the Failure terminal because of a problem that occurred in the Out or Catch flows (for example, repeated parsing errors that caused the backout threshold to be met), the original parsing errors are not in the exception list; the exception list contains the exception that indicates that the backout threshold has been met.
- The MQInput and TimeoutNotification nodes have additional processing for transactional messages (other input nodes do not handle transactional messages).

  For more information, see "Handling MQInput node errors" on page 1886 and "Handling timeout notification errors" on page 1889.
- If you include a Trace node that specifies `$Root` or `$Body`, the complete message is parsed. This might generate parser errors that are not otherwise detected.

The general principles of error handling are:

- If you connect the Catch terminal of the input node, you are indicating that the flow handles all the exceptions that are generated anywhere in the out flow. The integration node does not rollback a

transaction unless there is an exception on the catch flow. If the catch flow completes successfully, the message is committed. If you want any rollback action after an exception has been raised and caught, you must provide this in the catch flow.

- If you do not connect the Catch terminal of the input node, you can connect the Failure terminal and provide a fail flow to handle exceptions generated by the node. The fail flow is started immediately when an exception occurs in the node.

  The HTTPInput node does not propagate the message to the Failure terminal if an exception is generated beyond the node and you have not connected the Catch terminal.
- If a node propagates a message to a catch flow, and another exception occurs that returns control to the same message flow node again, the node handles the message as though the Catch terminal is not connected.
- If you do not connect either the Failure or Catch terminals of the input node, the integration node provides default processing (which varies with the type of input node).
- If you need a more comprehensive error and recovery approach, include one or more TryCatch nodes to provide more localized areas of error handling.
- If you have a common procedure for handling particular errors, you might find it appropriate to create a subflow that includes the sequence of nodes required. Include this subflow wherever you need that action to be taken.
- If you have set security processing in an input node, and a security exception is thrown, then the message is not propagated on the message flow. The message is either backed out, or an error is returned. For more information, see “Security exception processing” on page 2582.
- Parsing exceptions are passed straight to the Failure terminal. If the Failure terminal is not connected, or an exception occurs in the fail flow, then the message is backed out. The backouts in the fail flow continue until the backout threshold has been exceeded once, and not twice as before.

For more information, see “Connecting Failure terminals” on page 1885, and “Catching exceptions in a TryCatch node” on page 1890.

If your message flows include database updates, the way in which you configure the nodes that interact with those databases can also affect the way that errors are handled:

- You can specify whether updates are committed or rolled back. You can set the node property `Transaction` to specify whether database updates are committed or rolled back with the message flow (option Automatic), or are committed or rolled back when the node itself terminates (option Commit). You must ensure that the combination of these property settings and the message flow error processing give the correct result.
- You can specify how database errors are handled. You can set the properties `Treat warnings as errors` and `Throw exception on database error` to change the default behavior of database error handling.

Message flows for aggregation involve additional factors that are not discussed in this topic. For information about message flows for aggregation, see “Handling exceptions in aggregation flows” on page 1837.

Connecting Failure terminals

When a message flow node that has a Failure terminal detects an internal error, it propagates the message to that terminal. If it does not have a Failure terminal, or the Failure terminal is not connected, the integration node generates an exception.

About this task

Message flow nodes can generate errors that you can predict. In these cases, you might want to consider connecting the Failure terminal to a sequence of nodes that can respond to the expected errors.

Some examples of expected errors:

- Temporary errors when the input node retrieves the message.

- Validation errors that are detected by an MQInput, Compute, or Mapping node.
- Messages with an internal or format error that cannot be recognized or processed by the input node.
- Acceptable errors when a node accesses a database, and you choose not to configure the node to handle those errors.
- ESQL errors during message flow development. Some ESQL errors cannot be detected by the editor, but are recognized only by the integration node; these errors cause an exception if the Failure terminal is not connected. You can then remove the fail flow when you finish testing the runtime ESQL code.

You can also connect the Failure terminal if you do not want IBM MQ to try a message again, or put it to a backout or dead letter queue. For more information, see "Handling MQInput node errors" on page 1886.

Handling MQInput node errors

The MQInput node has its own error handling for message flows. Consider how you want your solution to handle errors when your message flow contains an MQInput node.

Warning: When you set up your IBM MQ queue manager, configure a backout queue, or a dead letter queue (DLQ), or both, to ensure that failure errors are handled correctly. If you do not configure a backout queue or DLQ, you might risk message loss, as described in "Handling rollbacks and then attempting to process the message again" on page 1887. If your solution requires that you do not use either of these queues, consider how you will handle failure errors in your message flow design.

Internal MQInput node error conditions

The MQInput node handles errors as described in "Handling errors in message flows" on page 1883. In addition, the MQInput node detects an internal error in the following situations:

- The queue manager that is specified on the MQInput node cannot be reached. For example, it might be stopped, or not yet created. The message flow that contains the MQInput node deploys, but an error shows in the syslog that messages cannot be read from the specified queue. When the queue manager becomes available, the MQInput node automatically starts reading messages from the queue. This error also occurs if the queue manager is available, but the specified queue is unavailable.
- If a connection to a queue manager is lost, and that connection is enlisted in a transaction, automatic reconnection is delayed until the inflight transaction is complete. Other queue managers that are required, but are not part of the transaction, reconnect automatically without delay. A completed transaction can include the following cases:
  - The unavailable MQ resource was not required and was not used, because exception handling was defined in the message flow, so the inflight transaction was successful and committed.
  - The unavailable MQ resource was required, and the message flow cannot succeed without it, so the inflight transaction was rolled back.
- An exception occurs when the associated message parser is initialized. If the `Parse timing` property is set to `Immediate` or `Complete`, the parser parses the input message after initialization. This parsing can cause a parsing or validation error, which is treated as an internal error.
- A warning is received on an `MQGET` call. This condition is handled in the same way as a parsing error.
- If an error occurs in a transactional message flow, and a Failure terminal is not connected:
  1. The message is rolled back to the input queue.
  2. The MQInput node writes the error and MQPUT error to the local error log, and starts its retry logic, as described in "Handling rollbacks and then attempting to process the message again" on page 1887.
- When a message is rolled back to the input queue, and the IBM MQ backout threshold is reached. This case is described in "Handling rollbacks and then attempting to process the message again" on page 1887.

MQ connection errors

The MQInput node attempts to connect to the queue manager when the flow is deployed and started. The MQOutput, MQGet, and MQReply nodes attempt to connect when the first message is sent or received. If any connection problems occur, see the IBM MQ product documentation for information about any `mqrc` return code values that are reported in the IBM App Connect Enterprise BIP messages.

Handling rollbacks and then attempting to process the message again

When a message is rolled back to the input queue, the MQInput node attempts to process the message again. The MQInput node checks whether the backout threshold value has been reached. The `BackoutCount` for each message is maintained by IBM MQ in the MQMD.

When you create the queue, you can specify the value of the backout threshold attribute BOTHRESH, or use the default, 0. If you accept the default value of 0, the MQInput node increases this value to 1. The MQInput node also sets the value to 1 if it cannot detect the current value. If the backout threshold is set to 1, the message is processed once, and does not try again through the Out terminal of the MQInput node. For the message to try again at least once, set the backout threshold to 2.

The retry behavior for the MQInput node is as follows:

1. The MQInput node reads the message, and checks the `BackoutCount` against the backout threshold.
2. If the backout threshold is not reached, the MQInput node attempts to get the message from the queue again. If the attempt fails, it is handled as an internal error. If the message processing succeeds on the Out terminal, the MQInput node propagates the message to the output flow.

   Typically, the backed out message is read again by the same MQInput node. However, depending on your IBM MQ queue manager configuration, it is possible for an MQInput node in a different message flow, or another IBM MQ application, to read the same input queue and get the backed-out message.

   If you want to control or prevent roll back, you can use the Catch terminal, as described in "Handling errors in message flows" on page 1883.
3. If the backout threshold is reached:
   - If a Failure terminal is connected, the MQInput node propagates the message to the fail flow. You must handle the error on the fail flow that is connected to the Failure terminal.

     When a message is propagated through the Failure terminal, the exception list does not contain the exceptions that occurred in the flows that are connected to the Out or Catch terminals. The exception list contains new exceptions only, which describe the reason that the message went through the Failure terminal (for example, that the backout threshold was reached).

     The MQInput node re-reads the message from the input queue before it is propagated through the Failure terminal. Therefore, the exception list does not contain the exceptions that occurred in the flows that are connected to the Out or Catch terminals. The exception list contains new exceptions only, which describe the reason that the message went through the Failure terminal (for example, that the backout threshold was reached).
   - If a Failure terminal is not connected, the MQInput node attempts to put the message on an available queue, in order of preference:
     a. If a backout queue name is defined for the input queue (queue attribute BOQNAME), the message is put on the backout queue.
     b. If the backout queue is not defined, or it cannot be identified by the MQInput node, the message is put on the dead letter queue (DLQ), if a DLQ is defined. The MQDLH **`PutApplName`** property

is set to `WebSphereMQIntegrator` and appended with the broker major version number, for example: `WebSphereMQIntegrator9`

c. If the message cannot be put on either of these queues because there is an MQPUT error (because the queue does not exist, or the queues cannot be identified by the MQInput node), the message cannot be handled safely without risk of loss.

The message cannot be discarded, therefore the message flow continues to attempt to backout the message. It records the error situation by writing errors to the local error log. A second indication of this error is the continual increase of the `BackoutCount` value of the message in the input queue.

If this situation occurs because neither queue exists, you can define one of the queues to resolve the problem. If the condition that prevented the message from being processed is resolved, you can temporarily increase the value of the B0THRESH attribute, which forces the message through normal processing.

4. If the backout threshold value is reached twice in the Failure terminal, the MQInput node attempts to put the message on an available queue, in the order of preference that is defined in step 2.

When a backed out message is read, the MQInput node can put the message to the backout queue or the DLQ. The `BackoutCount` value is checked when the message is received from the input queue. If the backout threshold is exceeded, the message is backed out immediately with no further processing performed on the message. The backout operation occurs in a separate transaction from previous processing failures, and the message is not parsed or validated during the backout transaction. The backout transaction generates its own set of monitoring events. Therefore, information that is obtained through message parsing, such as the *exceptionList*, might not be available.

Handling message group errors

IBM MQ supports message groups. You can specify that a message belongs to a group and its processing is then completed with the other messages in the group (that is, either all messages are committed or all messages are rolled back). When you send grouped messages to an integration node, this condition is upheld if the message flow is configured correctly, and errors do not occur during group message processing.

To configure the message flow to handle grouped messages correctly, review "Receiving messages in an IBM MQ message group" on page 783. However, if an error occurs while one of the messages is being processed, the message group might not be processed correctly.

If the MQInput node is configured as described, all messages in the group are processed in a single unit of work, which is committed when the last message in the group is successfully processed. However, if an error occurs before the last message in the group is processed, the unit of work that includes the messages, up to and including the message that generates the error, is subject to the error handling defined by the rules that are documented here, which might result in the unit of work being backed out.

Any of the remaining messages in the group might be successfully read and processed by the message flow, and therefore are handled and committed in a new unit of work. The unit of work is committed when the last message is processed. Therefore, if an error occurs in a group, but not on the first or last message, it is possible that part of the group is backed out and another part is committed.

Depending on the type of message processing that you require, you might need to provide extra error handling for message groups. For example, you might record the failure of the message group in a database, and include a check on the database when you retrieve each message. This check would force a rollback if the current group already encountered an error. This error handling configuration would ensure that the whole group of messages is backed out and not processed unless all are successful.

Handling timeout notification errors

The TimeoutNotification node takes various actions when it handles errors with transactional messages, depending on whether the Failure and Catch terminals are connected.

- If the TimeoutNotification node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal:
    1. The TimeoutNotification node writes the error to the local error log.
    2. The TimeoutNotification node repeatedly tries to process the request until the problem has been resolved.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The integration node creates an exception list to represent the error, which is propagated to the Failure terminal as part of the message tree. The TimeoutNotification node and the integration node do not provide further failure processing. The message is written to the Failure terminal as part of the same transaction, and if the failure flow handles the error successfully the transaction is committed.
- If the TimeoutNotification node has successfully propagated the message to the Out terminal and an exception is thrown in the flow connected to the Out terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and takes one of the steps:
  - If you have not connected the Catch terminal, the TimeoutNotification node tries to process the message again until the problem is resolved.
  - If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The integration node creates an exception list to represent the error, which is propagated to the Catch terminal as part of the message tree. The TimeoutNotification node and the integration node do not provide further failure processing. The message is written to the Catch terminal as part of the same transaction, and if the flow connected to the Catch terminal handles the error successfully the transaction is committed.
- If the TimeoutNotification node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log and tries to process the message again.
- If the TimeoutNotification node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the TimeoutNotification node. The TimeoutNotification node writes the error to the local error log, and tries to process the message again. The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the following table.

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
The node detects an internal error.	The flow that is connected to the Failure terminal handles the error.	The error is logged and the node retries the operation.	Not applicable	Not applicable
The node propagates a message to the Out terminal and an exception occurs in the output flow.	Not applicable	Not applicable	The flow that is connected to the Catch terminal handles the error.	The error is logged and the node retries the operation.

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
The node propagates a message to the Catch terminal and an exception occurs in the flow that is connected to the Catch terminal.	The error is logged and the node retries the operation.	The error is logged and the node retries the operation.	Not applicable	Not applicable
The node propagates a message to the Failure terminal and an exception occurs in the flow that is connected to the Failure terminal.	Not applicable	Not applicable	The error is logged and the node retries the operation.	The error is logged and the node retries the operation.

Catching exceptions in a TryCatch node

You can design a message flow to catch exceptions before they are returned to the input node. You can include one or more TryCatch nodes in a flow to provide a single point of failure for a sequence of nodes. With this technique, you can provide specific error processing and recovery.

About this task

A TryCatch node does not process a message in any way, it represents only a decision point in a message flow. When the TryCatch node receives a message, it propagates it to the Try terminal. Control is passed to the sequence of nodes that are connected to that terminal (the try flow).

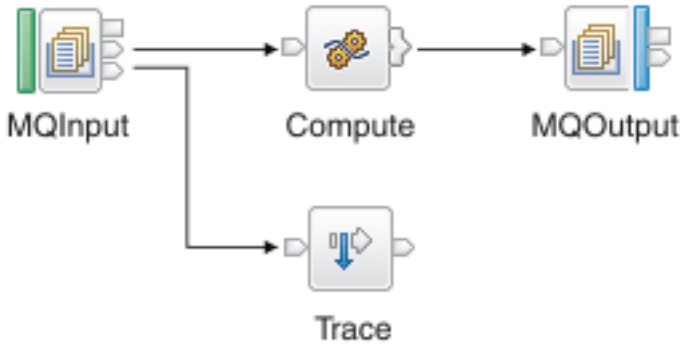
If an exception is thrown in the try flow,control is returned to the TryCatch node. The TryCatch node writes the current contents of the exception list tree to the local error log, then writes the information for the current exception to the exception list tree, overwriting the information that is stored there.

The TryCatch node propagates the message to the sequence of nodes that are connected to the Catch terminal (the catch flow). The content of the message tree that is propagated is identical to the content that was propagated to the Try terminal, which is the content of the tree when the TryCatch node first received it. The node enhances the message tree with the new exception information that it has written to the exception list tree. Any modifications or additions that the nodes in the try flow made to the message tree are not present in the message tree that is propagated to the catch flow.

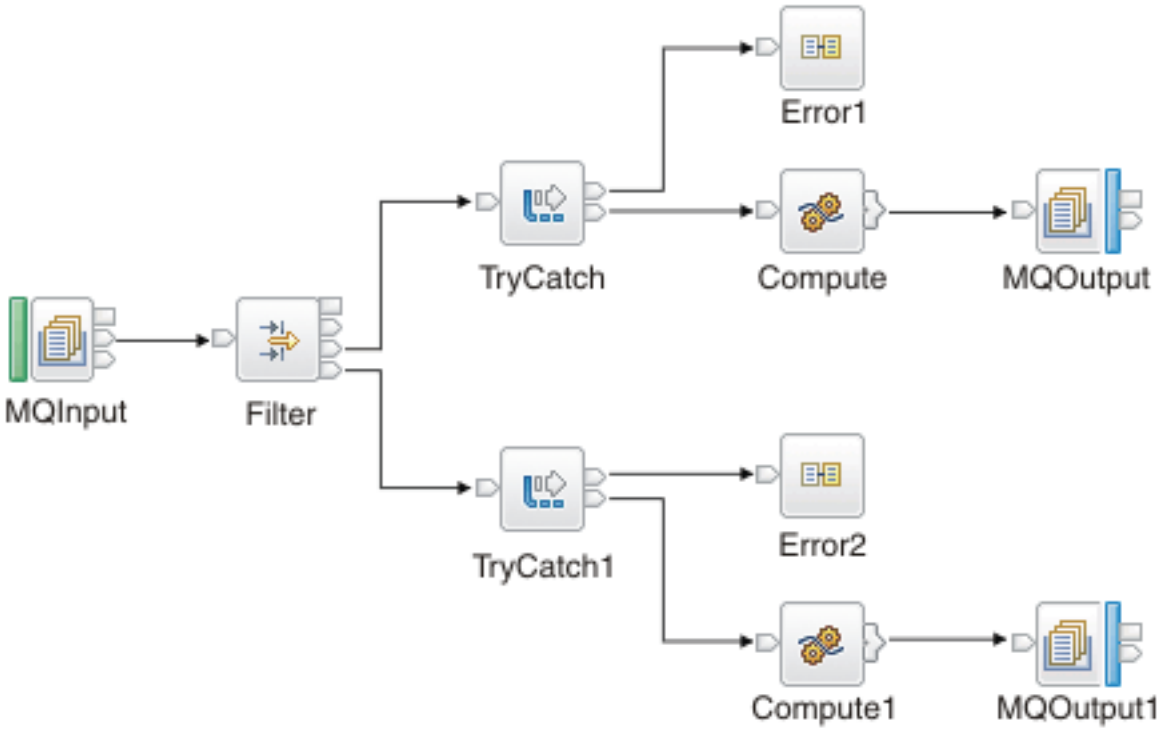
However, if the try flow completes processing that involves updates to external databases, these updates are not lost; they persist while the message is processed by the catch flow, and the decision about whether the updates are committed or rolled back is made on the configuration of your message flow and the individual message flow nodes that interact with the databases. If the updates are committed because of the configuration that you set, you must include logic in your catch flow that rolls back the changes that were made.

If an exception is thrown in the catch flow of the TryCatch node (for example, if you include a Throw node, or code an ESQL THROW statement), then control is returned to the next catch point in the message flow, such as another TryCatch node.

The following example shows how you can configure the flow to catch exceptions in the input node. The MQInput node's Catch terminal is connected to a Trace node to record the error.



In the following example, the message flow has two separate processing flows connected to the Filter node's True and False terminals. Here a TryCatch node is included on each of the two routes that the message can take. The Catch terminal of both TryCatch nodes is connected to a common error processing subflow.



If the input node in your message flow does not have a Catch terminal, and you want to process errors in the flow, you must include a TryCatch node.

Developing integration tests

You can create and run unit tests to validate the operation of your IBM App Connect Enterprise flows and message flow nodes.

Before you begin

Read the following topics:

- "Integration testing overview" on page 1893

About this task

You can create and run unit tests while you are developing your message flows and nodes, or you can create and run tests for flows that have already been deployed, by generating tests in bulk from messages that were recorded as they passed through a message flow.

The tests that you use to validate the message flows and nodes are written in Java and are stored in App Connect Enterprise test projects. You can write the tests by using a text editor, a source code editor, or a Java IDE such as the IBM App Connect Enterprise Toolkit. The simplest way to create a unit test for a message flow node is to use the **Create Test Case** wizard in the Toolkit, as described in "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894. For information about generating tests from recorded messages, see "Generating tests from recorded messages in a message flow" on page 1908.

To test a message flow node, an input message assembly is required to invoke the node, and the contents of the output message assembly are examined to determine whether the operation of the message flow node was successful. A message assembly is an object that is used to represent an instance of a message at a given point in time as it is being processed by a message flow. The message assembly contains the message headers and payload (which can be XML, JSON, DFDL or BLOB data), together with the environment tree, the local environment tree, and the exception list tree. The contents of the message assembly are accessed and modified by each message flow node as the message is processed.

A set of Java classes in the supplied `IntegrationTest.jar` file provides a mechanism for creating, building, testing and comparing message assemblies and their contents.

You can record messages as they pass through a message flow, and each recorded message is stored as an `.mxml` file, which is a serialized form of a message assembly. These `.mxml` files can then be used by unit tests, to validate the operation of the message flow nodes and flows. You can record the messages and create the serialized message assembly files either by setting the **`RecordAllMessages`** property in the integration server's `server.conf.yaml` file, or by opening the flow in the IBM App Connect Enterprise Toolkit and invoking the **Create Test Case** wizard on the message flow node in the Flow Exerciser.

You can control the level of granularity in your testing by creating integration tests for a single message flow node, a sequence of message flow nodes (such as a subflow or callable flow), or a whole message flow:

- **Testing a message flow node:**

  You can test the operation of an individual node in a message flow, in isolation from the rest of the nodes in the flow. An input message is passed into the message flow node, and the operation of the node is validated by comparing the output message against the expected output, together with any secondary actions such as updates to a database or any exceptions thrown. The tests pass messages directly to the message flow node that is being tested, rather than through the input node of the flow; this means that the owning message flow must be deployed to the integration server and initialized, but it does not need to be running.

- **Testing a partial message flow:**

  You can test a subflow or sequence of nodes in a message flow without testing the whole flow. An input message is passed into the first message flow node in the sequence and it is propagated through the flow until it reaches a defined stop point or the end of the flow. The tests pass messages directly to the first message flow node in the sequence, rather than through the input node of the flow; this means that the message flow being tested must be deployed to the integration server, but it does not need to be running.

  The success of the test can be established by validating the route of the message as it propagates through the flow, and the contents of the message at any point throughout the flow, as well as any secondary actions such as updates to a database or exceptions thrown.

  When testing a sequence of node nodes, you might need to alter the behavior of some nodes in the sequence to avoid accessing unavailable resources or to force the execution to follow a particular path.

- **Testing a whole message flow:**

  You can test a whole message flow by invoking the message flow using its published interfaces, such as HTTP, MQ, File, and so on. The input nodes are typically invoked using their native transport interfaces supported by the input node called from the code in the test project. The message flow is invoked through the input node, which means that the message flow must be deployed and running in order for the tests to run.

  Like partial message flow testing, the test can validate the route of the message as it propagates through the flow, in addition to the contents of the message at any point throughout the flow. The test can also alter the behavior of nodes in the flow by substituting responses or by modifying behavior.

For information that you need to know before you start developing integration tests, see "Integration testing overview" on page 1893.

Procedure

Read the following sections to learn more about developing and running integration tests:

- "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894
- "Generating tests from recorded messages in a message flow" on page 1908
- "Testing a sequence of message flow nodes in a single test case" on page 1912
- "Editing a message assembly" on page 1921
- "Running integration tests" on page 1932
- "Example message flow node tests " on page 1938

For demonstrations of some of the App Connect Enterprise integration testing capabilities, see the following videos:

- Creating and running a JUnit test for a message flow node
- An in-depth look at how to create a test case for a message flow node
- Understanding a message assembly
- Generating a test case using recorded messages in the Flow Exerciser
- Generating a test case for a REST API operation
- Creating a test case for a message flow that uses external resources
- Generating and running tests from the command line

Integration testing overview

You can create and run unit tests to validate the operation of your IBM App Connect Enterprise flows and message flow nodes.

IBM App Connect Enterprise facilitates test-driven development through its test framework, which makes it easier to adopt new product versions and make architectural changes, such as modernizing and moving to a container-based environment. You can use integration tests to check whether your integration flows operate as expected after development changes, upgrades, or changes to services that your flow interacts with. You might also want to check whether changes to external services have impacted the operation of your flows. By using this integration test capability as part of your test-driven development, you can develop and run unit tests more rapidly as part of a pipeline, without the need for third-party tools.

In addition to the test framework for running unit tests in the integration server process, the IBM App Connect Enterprise Toolkit provides powerful tools to help you write the tests, including a template that you can use as a starting point for developing a variety of tests. You can develop spies that spy on the behavior of an individual message flow node, or stubs that describe how you can interact using trained messages to define the behavior of a particular part of the flow. If your flow invokes an external service or a database, you might not have access to that system when you are executing the tests; in this situation it can be useful to record messages and then use them as part of your unit tests, without requiring the external services to be online.

Figure 5. Unit test using trained messages in place of an offline system

App Connect Enterprise also enables interaction with third-party matcher libraries. Support is provided for JUnit assertions and third-party matcher libraries such as Hamcrest, JSONAssert, and XMLUnit, as well as matchers for the App Connect Enterprise message assembly.

In addition to creating and running tests during development, you can test existing flows that have already been deployed, by recording messages in bulk and then using them to generate tests for each of the message flow nodes. To automatically create tests for deployed flows, you switch on recording for the flows, store the messages that flow through them, and then generate tests from those stored message assemblies (`.mxml files`) by using the **`ibmint generate tests`** command. You can extend these generated tests with additional functionality, or you can use the stored message assemblies to write your own tests.

For information about how to develop unit tests for your message flows and nodes, see "Developing integration tests" on page 1891.

Developing integration tests by using the IBM App Connect Enterprise Toolkit

You can create a test case for a message flow node by using a wizard in the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- "Integration testing overview" on page 1893
- "Developing integration tests" on page 1891

About this task

You can use the IBM App Connect Enterprise Toolkit to create a test case for a message flow node, either from a recorded message flow or by using XML or JSON files to generate input or output message assembly files. You can also create a test case for a REST API operation subflow or for an individual message flow node in a REST API operation subflow.

For information about how to record messages and then use them to generate tests, see "Generating tests from recorded messages in a message flow" on page 1908.

Procedure

Use the Toolkit to create a test case for an individual message flow node or REST API subflow, by completing the steps in one of the following topics:

- "Creating a test case for a message flow node using recorded messages" on page 1895
- "Creating a test case using XML or JSON data files" on page 1900
- "Creating a test case for a REST API operation subflow" on page 1902
- "Creating a test case for a message flow node in a REST API operation subflow" on page 1904

Creating a test case for a message flow node using recorded messages

You can use the IBM App Connect Enterprise Toolkit to create a test case for a message flow node by using messages that are recorded as they pass through a message flow.

Before you begin

Read the following topics:

- "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894
- "Integration testing overview" on page 1893

About this task

The steps in this task include importing and running a tutorial to create a message flow and record messages as they pass through the flow, and then using the recorded messages to generate unit tests. In this example, the tests are generated for a Compute node in a simple message flow.

Video: Creating and running a JUnit test for a message flow node

This video demonstrates the capability to create a JUnit Test for a message flow node using recorded messages in the Flow Exerciser.

For more videos about generating and running tests, see the IBM App Connect Enterprise playlist on YouTube.

Procedure

Complete the following steps to create a test case for a message flow node by using messages in a recorded message flow:

1. In the IBM App Connect Enterprise Toolkit, open the Tutorials Gallery and select **Using an HTTP input to drive a message flow**.
2. Follow the instructions in the tutorial.

   The tutorial provides a simple message flow consisting of HTTP input and reply nodes, and a Compute node to convert two field names. The tutorial then guides you through using the Flow Exerciser to record the flow of messages through the message flow.

When messages have been sent through the flow, you can create a test case using the messages that were recorded by the Flow Exerciser.

In this example, an input message assembly is used for input to the Compute node and an output message assembly is propagated from the Compute node, as shown in the following diagram:

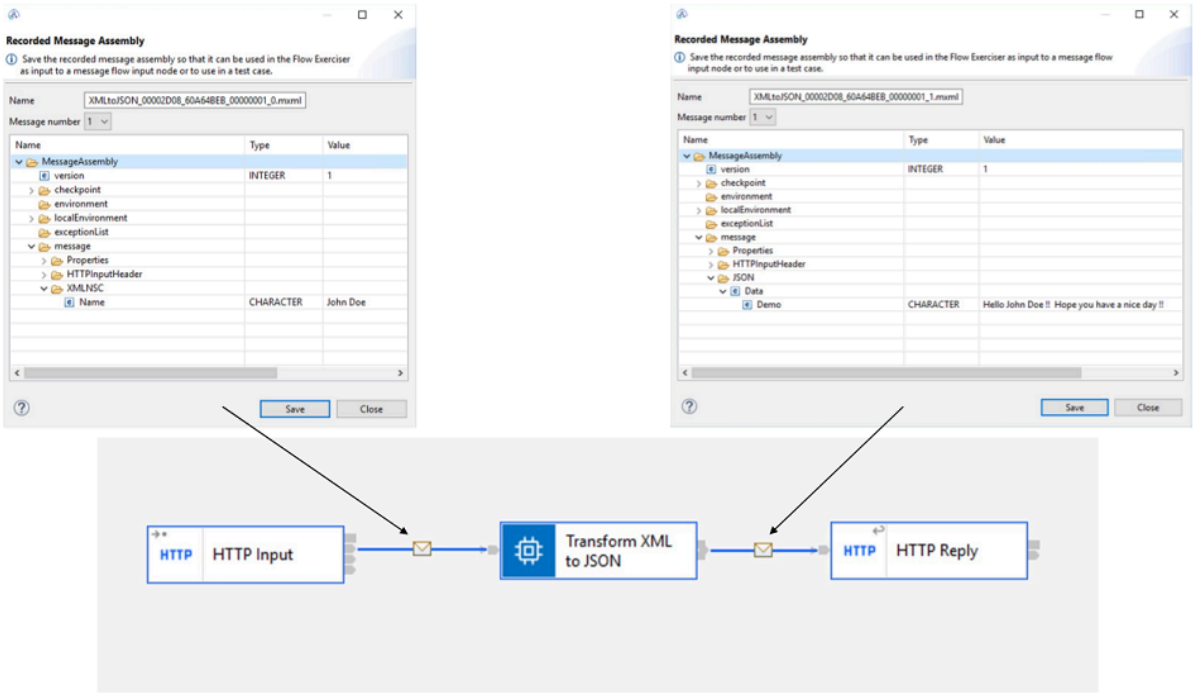


Figure 6. Message assemblies used as input and output for a Compute node test.

You can then use these message assemblies to create a test case for testing the Compute node.

3. In the Flow Exerciser, right-click the Compute node in the message flow and click **Create Test Case...**.

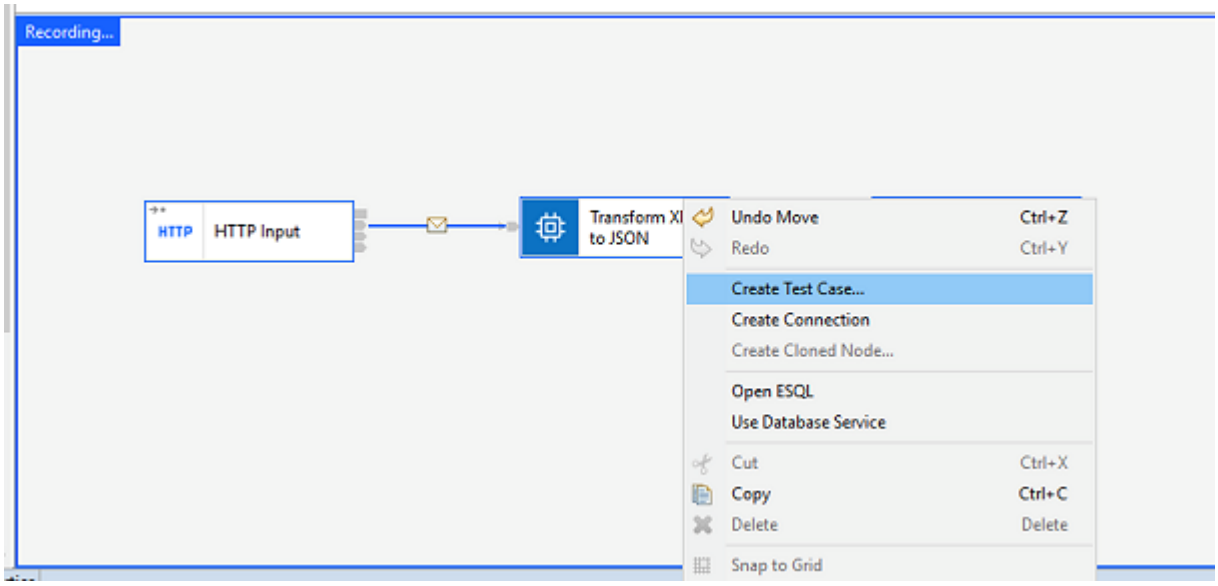


Figure 7. Selecting the Create Test Case option for a Compute node test

4. The **Create Test Case** dialog is displayed, containing information about the selected message flow node, the names of the files to be used for the input and output message assemblies, the assertions

and matchers to be included in the generated test, and a read-only preview of the test case that will be generated for the message flow node.

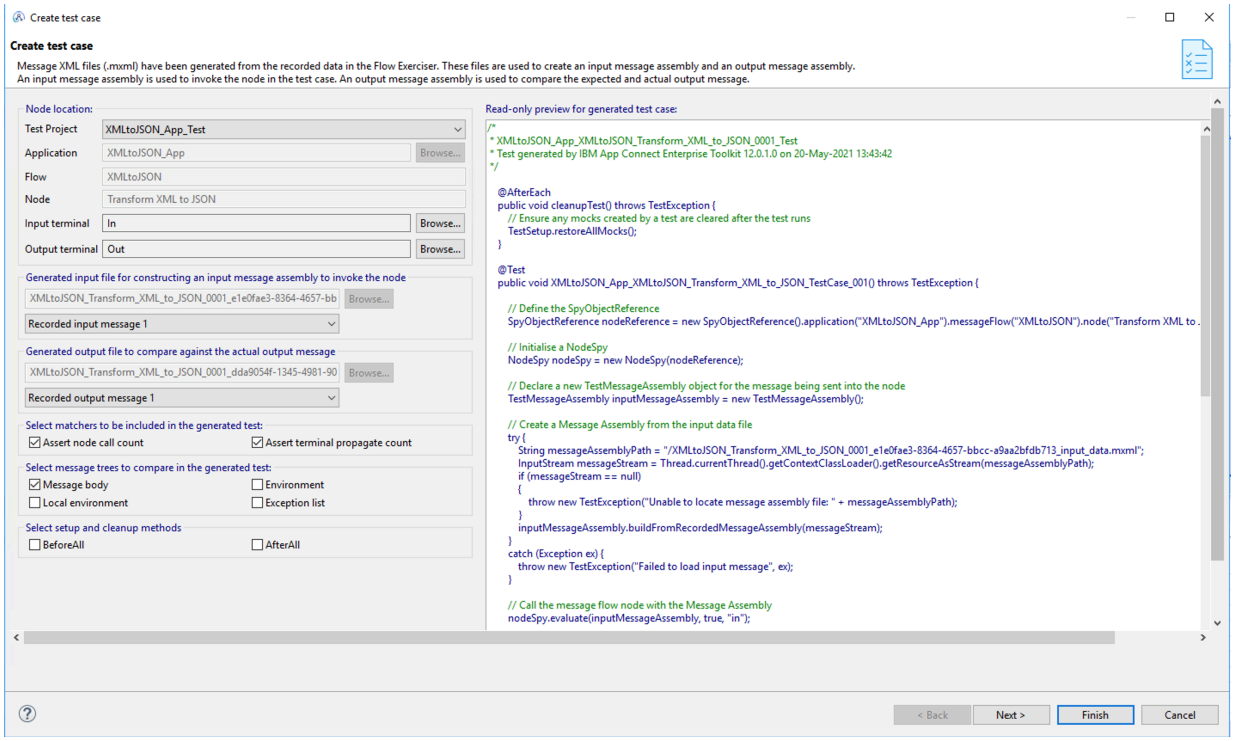


Figure 8. Create Test Case dialog for a message flow node test

5. You can choose which message trees in the message assembly are to be compared in the generated test, by selecting one or more of the following options:
   - Message body
   - Local environment
   - Environment
   - Exception list

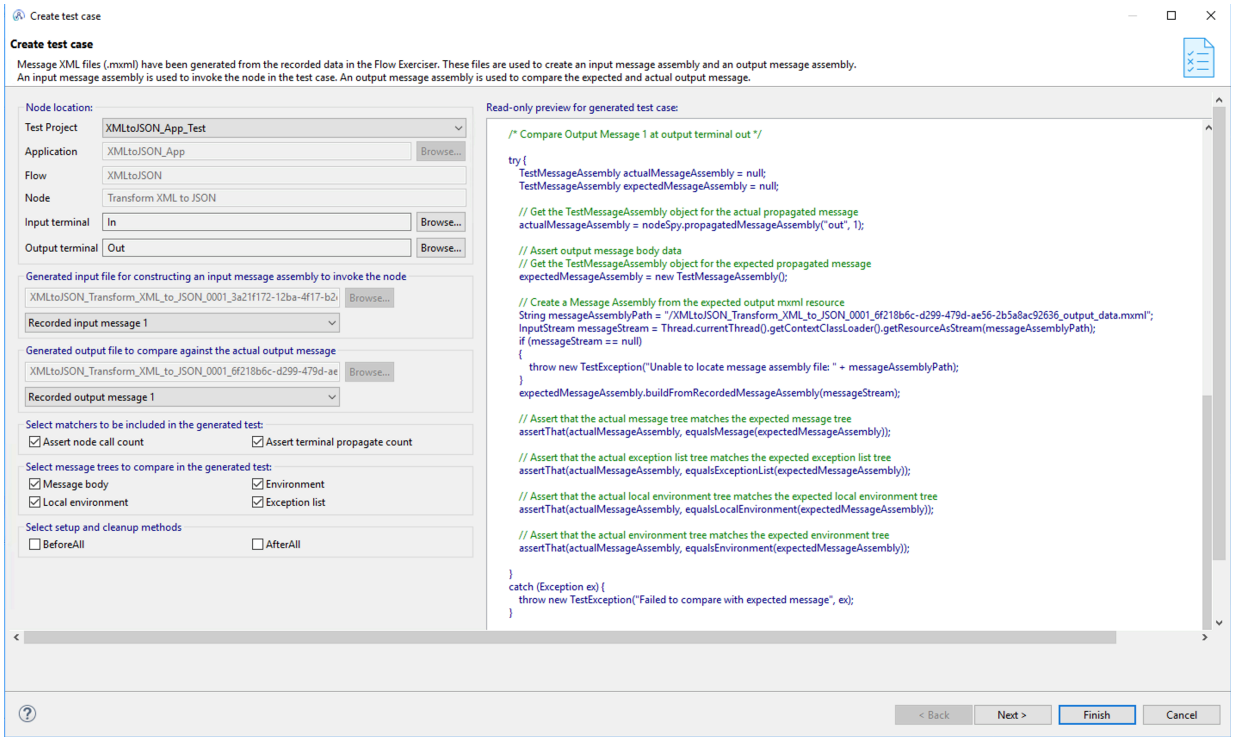


Figure 9. Select message trees to be compared

6. Click **Finish**.

   The generated test case is displayed in the Java Editor, showing the following objects:

   - The test project name: `XMLtoJSON_App_Test` (which is the *ApplicationName* followed by `_Test`)
   - The class name: `XMLtoJSON_App_XMLtoJSON_Transform_XML_to_JSON_0001_Test` (which is the *ApplicationName_FlowName_NodeName* followed by `0001_Test`)
   - The test case method name: `XMLtoJSON_App_XMLtoJSON_Transform_XML_to_JSON_TestCase_001()` (which is the *ApplicationName_FlowName_NodeName* followed by `TestCase_001()`)
   - The generated message assembly files (`*.mxml`) are located in the `resources` folder in the test project.

   Any character that is not valid for Java is changed to an underscore character (_). In this example, the spaces in the integration node name (`Transform XML to JSON`) are replaced with underscores in the generated class and method name (`Transform_XML_to_JSON`).

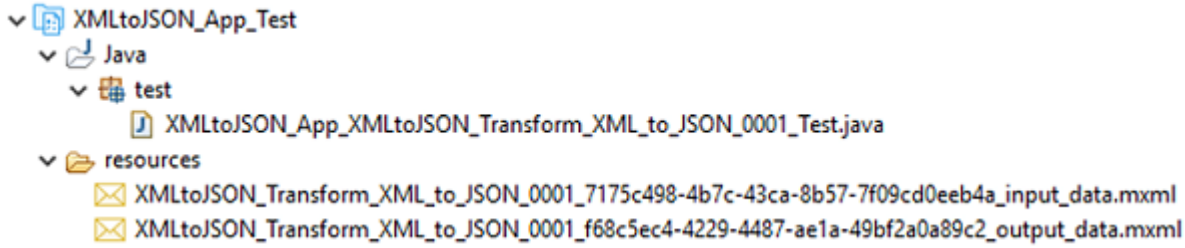


Figure 10. Objects generated by the Create Test Case wizard

7. The required objects are added to the Java test class automatically, and you can then edit the `.java` file to make additional modifications, if required. For example:

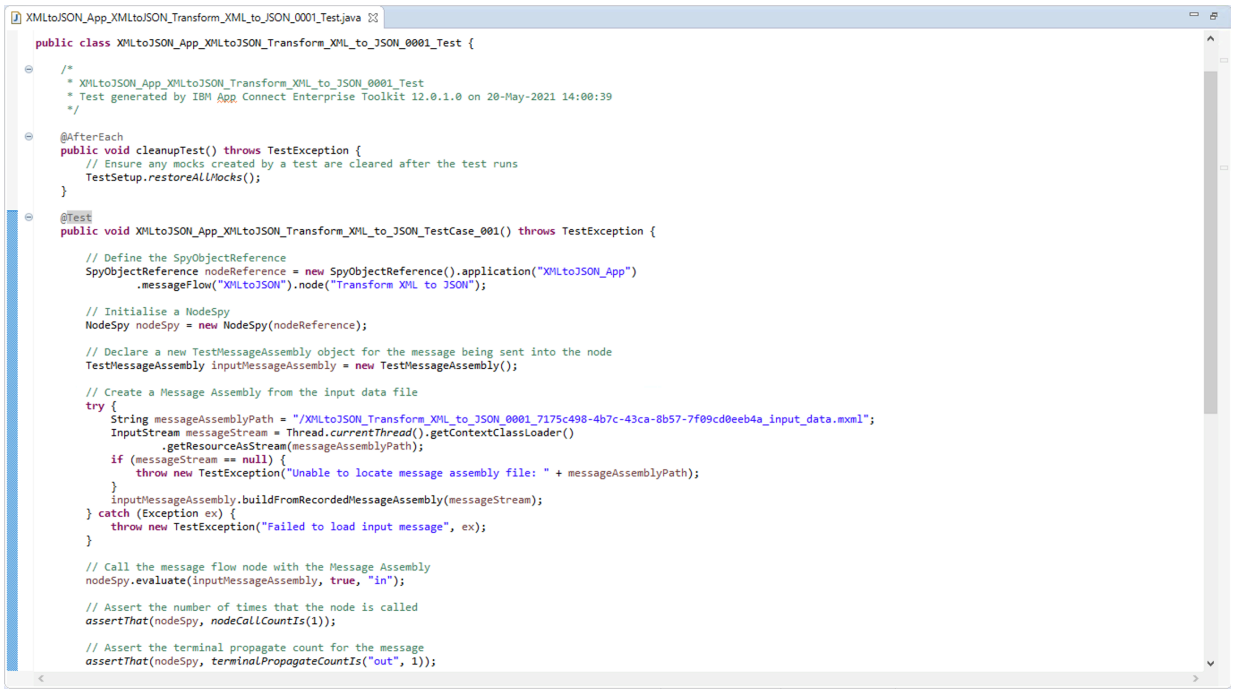


Figure 11. Java test class for Mapping node unit test

8. Optional: You can run the test by right-clicking the test project `XMLtoJSON_App_Test` and then clicking **Run Test Project**.

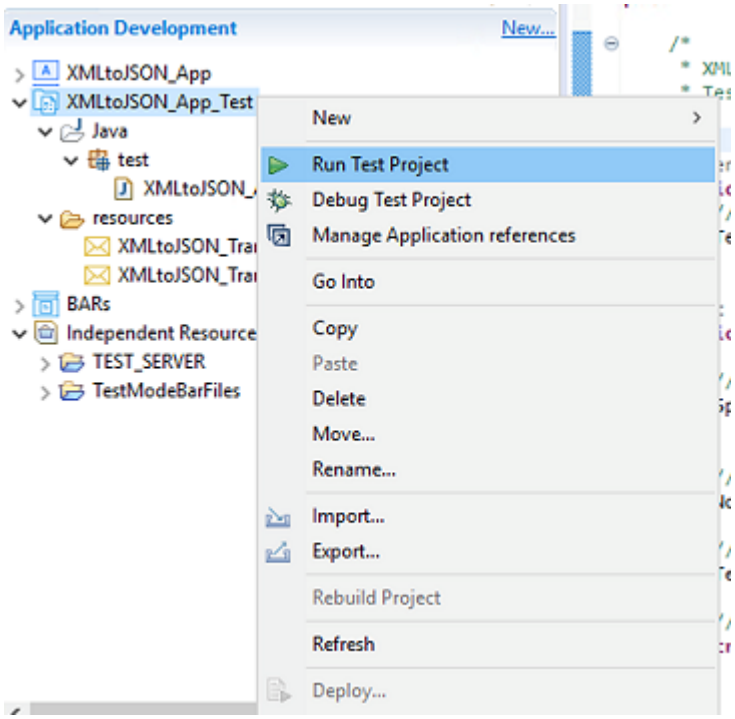


Figure 12. Run the test project

9. Optional: View the test result by opening the JUnit view in the Toolkit:

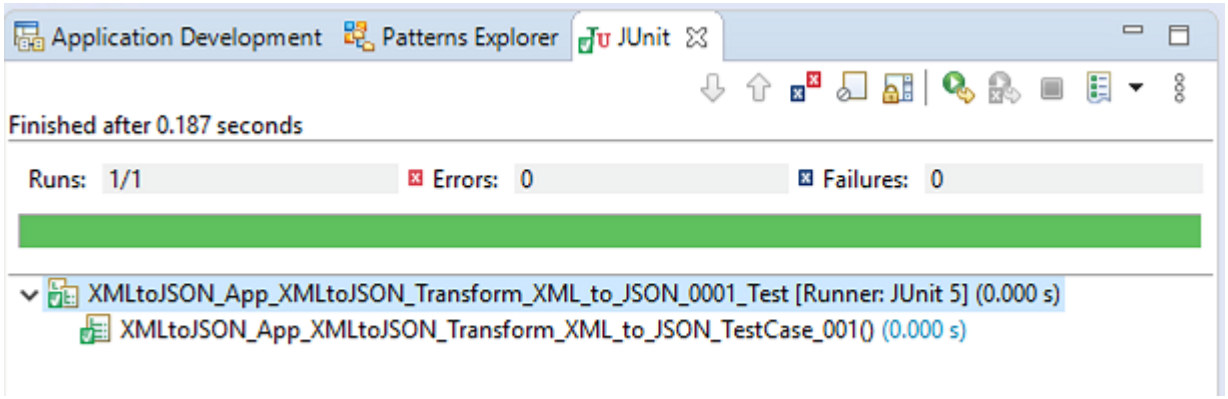


Figure 13. Test result in the JUnit view of the Toolkit

Creating a test case using XML or JSON data files

You can use the IBM App Connect Enterprise Toolkit to create a test case using XML or JSON files to generate input or output message assembly files.

Before you begin

Read the following topics:

- "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894
- "Integration testing overview" on page 1893

About this task

You can use the **Create Test Case** wizard in the IBM App Connect Enterprise Toolkit to select an input file and an output file to create input and output message assemblies. The example used in the following steps shows a Compute node being used to transform an XML message to a JSON message:

Procedure

1. In the Flow Exerciser, right-click the Compute node in your message flow and then select **Create Test Case**.

   The **Create test case** dialog is displayed, containing information about the selected message flow node, and a read-only preview of the test case that will be generated for the message flow node.

2. Select an XML file to be used as an input message assembly, and a JSON file to be used as an output message assembly. The files that you select must be in the toolkit workspace or filesystem. In the following example, the selected files are `demo_input.xml` and `demo_output.xml`:

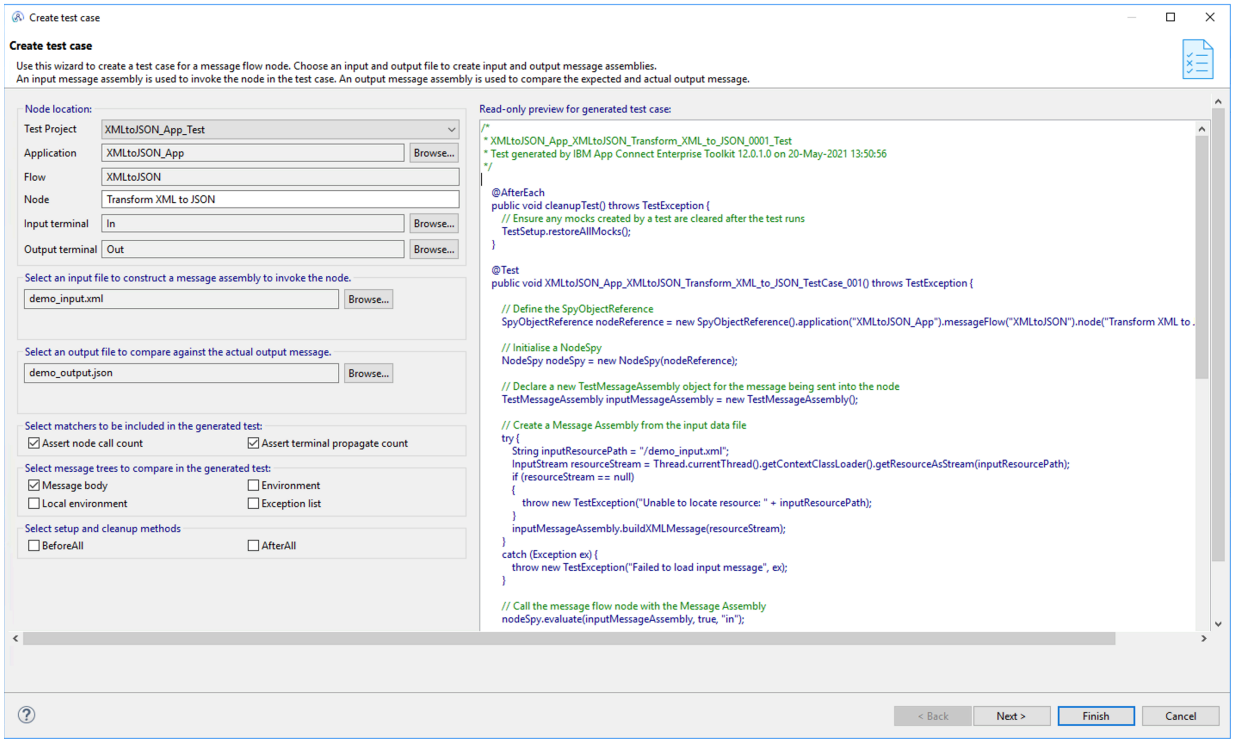


Figure 14. Create test case wizard

When an XML data file is used, the `TestMessageAssembly` object is created using the `buildMessageBodyFromXML` method in the generated code. When a JSON data file is used, the `TestMessageAssembly` object is created using the `buildMessageBodyFromJSON` method in the generated code.

3. You can control the content of the test case code that is generated, by selecting options in the **Create test case** dialog.

   For example:

   - Select **Assert node call count** to add an assert that uses the `nodeCallCountIs` matcher
   - Select **Assert terminal propagate** to add an assert that uses the `terminalPropagateCountIs` matcher

4. Click **Finish**.

Creating a test case for a REST API operation subflow

You can create and run a test case for a REST API operation subflow.

Before you begin

Read the following topics:

- “Developing integration tests by using the IBM App Connect Enterprise Toolkit” on page 1894
- “Integration testing overview” on page 1893

About this task

REST API operations are implemented by subflows, and the OpenAPI document for the REST API describes the input and output messages that are used by each operation.

For information about how to use the IBM App Connect Enterprise Toolkit to generate a test case for an individual message flow node in a REST API subflow, see “Creating a test case for a message flow node in a REST API operation subflow” on page 1904.

Video: Generating a test case for a REST API operation

This video demonstrates the capability to create a JUnit test for an operation in a REST API. The capability can be used on both REST APIs that are using Open API v2.0 or v3.0 documents which define the APIs.

For more videos about generating and running tests, see the IBM App Connect Enterprise playlist on YouTube.

To create a test case for a whole REST API operation subflow, complete the following steps:

Procedure

1. In the Application Development view, right-click the operation subflow and then click **Create Operation Test Case**.

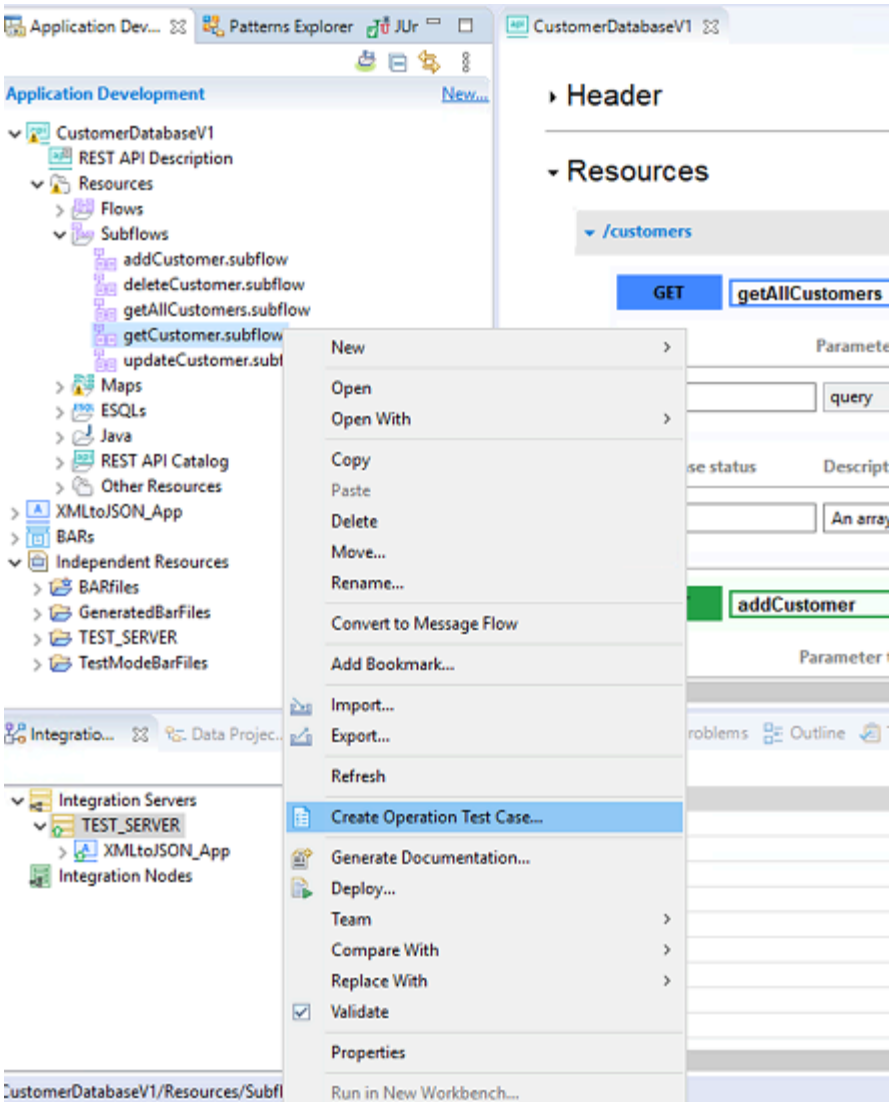


Figure 15. Create Operation Test Case option in the Application Development view

2. The **Create test case** wizard is displayed, containing a read-only preview of the generated test case.

   The OpenAPI document defines the message format used by each operation, and an instance of the message is generated and used to create the message assembly in the test case.

   Local environment entries for HTTP and REST are defined in the generated test case, using the path or query parameters for the operation. The input message assembly can be generated using the local environment entries.

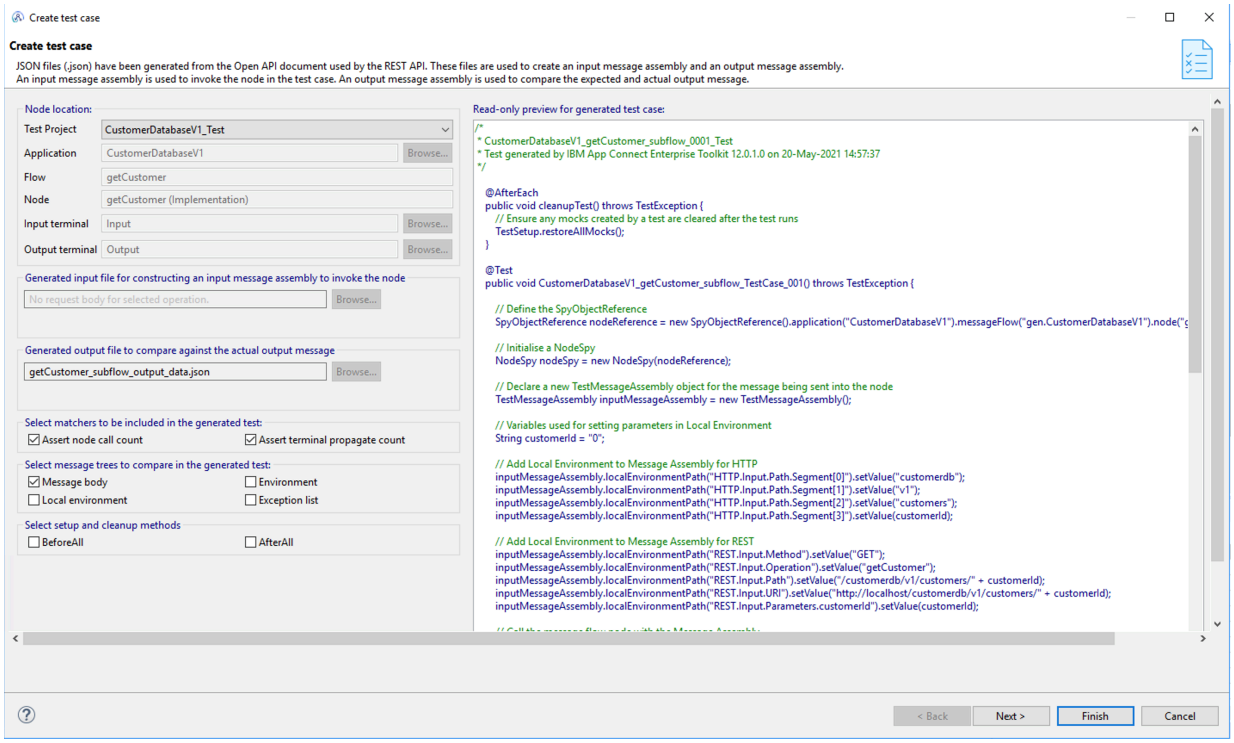


Figure 16. Create test case dialog for a REST API operation subflow

   For an example of a test case generated for a GET operation, see "Example test case for a REST API GET operation" on page 1906.

3. Click **Finish**.

Creating a test case for a message flow node in a REST API operation subflow

You can create and run a test case for an individual message flow node in a REST API operation subflow.

Before you begin

Read the following topics:

- "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894
- "Integration testing overview" on page 1893

About this task

REST API operations are implemented by subflows, and the OpenAPI document for the REST API describes the input and output messages that are used by each operation.

You can generate a test case for an individual message flow node in a REST API operation subflow, by following these steps:

Procedure

1. In the Application Development view of the Toolkit, open the REST API operation subflow that contains the node that you want to test, right-click the node, and then click **Create Test Case**.

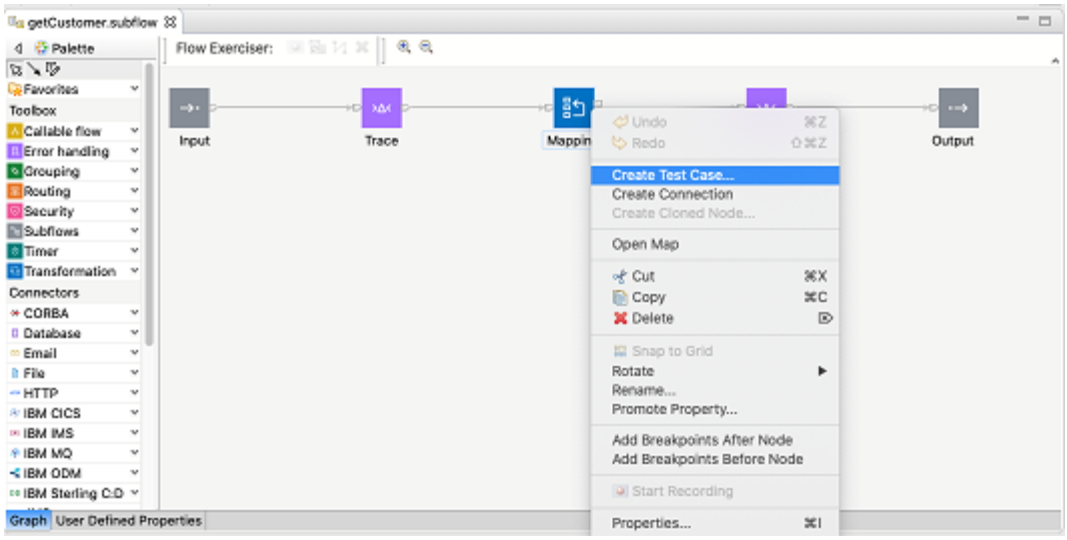


Figure 17. Create Test Case option in the Application Development view

2. The **Create test case** dialog is displayed, containing a read-only preview of the generated test case.

   The OpenAPI document defines the message format used by each operation, and an instance of the message is generated and used to create the message assembly in the test case.

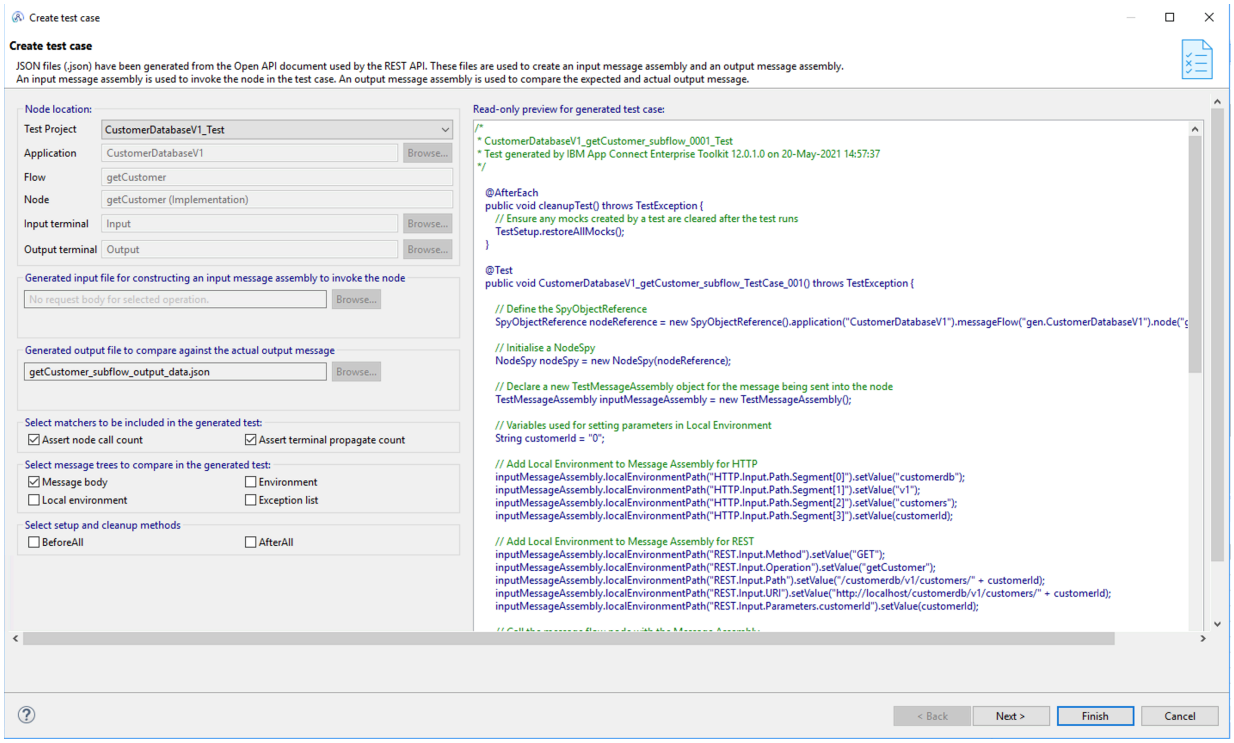


Figure 18. Create test case dialog for a REST API operation subflow

The node to be tested (in this case a Mapping node) is specified in the `SpyObjectReference` object in the generated test case, as shown in the following example:

```
// Define the SpyObjectReference
```

```
 SpyObjectReference nodeReference = new
 SpyObjectReference().application("CustomerDatabaseV1").messageFlow("gen.CustomerDatabaseV1").subflowNode(
 (Implementation)").node("Mapping");
```

If you generate a test case for a whole subflow, no node is specified in the SpyObjectReference object definition:

```
// Define the SpyObjectReference
 SpyObjectReference nodeReference = new
 SpyObjectReference().application("CustomerDatabaseV1").messageFlow("gen.CustomerDatabaseV1").node("getCus
 (Implementation)");
```

For an example of a test case generated for a GET operation, see "Example test case for a REST API GET operation" on page 1906.

3. Click **Finish**.

Example test case for a REST API GET operation

You can generate a test case for a REST API GET operation, as shown in this example.

Before you begin

Read the following topics:

- "Creating a test case for a REST API operation subflow" on page 1902

About this task

The following example code shows a test case generated for a GET operation, in which an ID is specified using a path parameter:

```
package test;

import java.io.InputStream;
import java.nio.charset.StandardCharsets;

import org.junit.jupiter.api.Test;

import com.ibm.integration.test.v1.NodeSpy;
import com.ibm.integration.test.v1.SpyObjectReference;
import com.ibm.integration.test.v1.TestMessageAssembly;
import com.ibm.integration.test.v1.exception.TestException;

import static com.ibm.integration.test.v1.Matchers.*;
import static net.javacrumbs.jsonunit.JsonMatchers.jsonEquals;
import static org.hamcrest.MatcherAssert.assertThat;

public class CustomerDatabaseV1_getCustomer_subflow_0001_IntegrationTest {

 /*
  * CustomerDatabaseV1_getCustomer_subflow_0001_IntegrationTest
  * Integration test generated by IBM App Connect Enterprise Toolkit 12.0.0.0 on 26-Apr-2021
 14:28:46
  */

 @Test
 public void CustomerDatabaseV1_getCustomer_subflow_TestCase_001() throws TestException {

  // Define the SpyObjectReference
  SpyObjectReference nodeReference = new SpyObjectReference().application("CustomerDatabaseV1")
    .messageFlow("gen.CustomerDatabaseV1").node("getCustomer (Implementation)");

  // Initialise a NodeSpy
  NodeSpy nodeSpy = new NodeSpy(nodeReference);

  // Declare a new TestMessageAssembly object for the message being sent into the node
  TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();

  // Variables used for setting parameters in Local Environment
  String customerId = "0";

  // Add Local Environment to Message Assembly for HTTP
```

```
 inputMessageAssembly.localEnvironmentPath("HTTP.Input.Path.Segment[0]").setValue("customerdb");
  inputMessageAssembly.localEnvironmentPath("HTTP.Input.Path.Segment[1]").setValue("v1");
  inputMessageAssembly.localEnvironmentPath("HTTP.Input.Path.Segment[2]").setValue("customers");
  inputMessageAssembly.localEnvironmentPath("HTTP.Input.Path.Segment[3]").setValue(customerId);

  // Add Local Environment to Message Assembly for REST
  inputMessageAssembly.localEnvironmentPath("REST.Input.Method").setValue("GET");
  inputMessageAssembly.localEnvironmentPath("REST.Input.Operation").setValue("getCustomer");
  inputMessageAssembly.localEnvironmentPath("REST.Input.Path").setValue("/customerdb/v1/
customers/" + customerId);
  inputMessageAssembly.localEnvironmentPath("REST.Input.URI")
    .setValue("http://localhost/customerdb/v1/customers/" + customerId);

 inputMessageAssembly.localEnvironmentPath("REST.Input.Parameters.customerId").setValue(customerId);

  // Call the message flow node with the Message Assembly
  nodeSpy.evaluate(inputMessageAssembly, true, "Input");

  // Assert the number of times that the node is called
  assertThat(nodeSpy, nodeCallCountIs(1));

  // Assert the terminal propagate count for the message
  assertThat(nodeSpy, terminalPropagateCountIs("Output", 1));

  /* Compare Output Message 1 at output terminal Output */

  try {

   TestMessageAssembly actualMessageAssembly = null;
   String actualOutputData = null;
   String expectedOutputData = null;

   // Get the TestMessageAssembly object for the actual propagated message
   actualMessageAssembly = nodeSpy.propagatedMessageAssembly("Output", 1);

   // Assert output message body data
   // Get the string containing the actual data that was propagated from the node
   actualOutputData = actualMessageAssembly.getMessageBodyAsString();

   // Create an InputStream from the expected resource
   String expectedResourcePath = "/getCustomer_subflow_output_data.json";
   InputStream resourceStream = Thread.currentThread().getContextClassLoader()
     .getResourceAsStream(expectedResourcePath);
   if (resourceStream == null) {
    throw new TestException("Unable to locate resource: " + expectedResourcePath);
   }
   byte[] buffer = new byte[resourceStream.available()];
   resourceStream.read(buffer);

   // Get the expected output data using output data file
   expectedOutputData = new String(buffer, StandardCharsets.UTF_8);

   // Assert that the actual data matches the expected data
   assertThat(actualOutputData, jsonEquals(expectedOutputData));

  } catch (Exception ex) {
   throw new TestException("Failed to compare with expected message", ex);
  }

 }

}
```

The output file `/getCustomer_subflow_output_data.json` is generated from the Open API definition for the REST API. It contains the following string:

```
{"id":0,"firstname":"string","lastname":"string","address":"string"}
```

The output message assembly is generated from `/getCustomer_subflow_output_data.json`. It is generated using the following code:

```
// Assert output message body data
// Get the string containing the actual data that was propagated from the node
 actualOutputData = actualMessageAssembly.getMessageBodyAsJSON();
```

Generating tests from recorded messages in a message flow

You can create and run unit tests of your message flows by recording messages that pass through a flow and then using the recorded message assemblies to generate the tests.

Before you begin

Read the following topics:

- "Developing integration tests" on page 1891
- "Integration testing overview" on page 1893

About this task

As an alternative to using the Toolkit to create and run tests during development, you can create tests using command line tools. You can configure an integration server to store message assembly files in a specified directory for every message that is processed by a message flow.

You can generate a test project by using the command line. The test project contains a test case for every node in a message flow, apart from an input node. Each test case uses a message assembly file for input to the node and another message assembly to do a comparison check.

Procedure

Complete the following steps to generate unit tests from recorded messages:

1. Configure an integration server to record messages
2. Generate a test project by using the ibmint generate tests command
3. Compile the test project and create a BAR file by using the command line
4. Optional: Import the test project into the toolkit for editing
5. Optional: Run the tests in the test project by using either the command line or the toolkit:
   - Run the tests in the test project by using the command line
   - Run the tests in the test project by using the toolkit

Configuring an integration server to record messages

You can configure an integration server to record message assembly files for all messages that are propagated through the message flow.

Before you begin

Read the following topics:

- "Generating tests from recorded messages in a message flow" on page 1908
- "Integration testing overview" on page 1893

About this task

The recorded message assembly files are XML files that have a `.mxml` file suffix. A recorded message assembly file is written each time a message flow node propagates a message from an output terminal to the input terminal of the next node in the message flow. You can import these files into a test project and use them in your tests as:

- Input data when invoking a message flow node
- Expected data for comparisons of the test results
- Substitute data to be returned by a Node Stub instead of invoking the actual node

You can configure an integration server to record all messages that pass through a message flow by setting properties in the **RecordedMessageManager** section of your integration server's `server.conf.yaml` file:

```
ResourceManagers:
   RecordedMessageManager:
      recordedMessagePath: 'C:\temp\IntegrationServer\recorded_messages'   # Set the directory
 to store recorded messages
      recordAllMessages: true  # Set to 'true' to enabling recording of messages from all
 message flows
```

Procedure

Complete the following steps to configure the recording of messages as they pass through a message flow:

1. Open the `server.conf.yaml` configuration file for your integration server by using a YAML editor.

   If you are not able to access a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not valid in YAML and will cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. Enable recording of all messages that pass through all message flows, by setting the **recordAllMessages** property to `true`.
3. Set the **recordedMessagePath** property to specify the location where the recorded message assembly files will be saved when recording is enabled.
4. Stop and restart the integration server for the changes to take effect.

   The recorded message assemblies are saved in `.mxml` files, in a subdirectory of the directory specified by the **recordedMessagePath** property.

Generating tests by using the `ibmint generate tests` command

You can use the **`ibmint generate tests`** command to generate a test project and unit tests by using recorded messages.

Before you begin

Read the following topics:

- "Generating tests from recorded messages in a message flow" on page 1908
- "Integration testing overview" on page 1893

About this task

The command recursively searches for recorded message assemblies in the specified directory name, and analyzes the files that are found, identifying pairs of messages that represent the corresponding input and output messages from the invocation of the message flow node. A Java test case is then created for each pair of messages, and is written to the test project directory together with the recorded message assembly files.

Procedure

1. Open a command console and run the **`ibmint generate tests`** command to generate tests based on the recorded message assembly (`.mxml`) files:

   ```
   ibmint generate tests
    --recorded-messages C:\tmp\IntegrationServer\recorded_messages
    --output-test-project C:\tmp\MyIntegrationTestProject
   ```

```
  --java-class com.ibm.mytests.MyIntegrationTestClass
```

where:

- **--recorded-messages** specifies the file name and path of the recorded messages that are to be used in the tests.
- **--output-test-project** specifies the name and location of the generated test project.
- **--java-class** specifies the fully-qualified classpath of the Java class that will contain the generated tests.

2. You can import the generated tests into the IBM App Connect Enterprise Toolkit for editing, as described in "Importing a test project into the IBM App Connect Enterprise Toolkit for editing" on page 1911, or you can build the tests by using Gradle before running them on an integration server, as described in "Compiling a test project and creating a BAR file by using the command line" on page 1910.

Compiling a test project and creating a BAR file by using the command line

You can compile a test project and create a BAR file by using the App Connect Enterprise command line.

Before you begin

Read the following topics:

- "Generating tests from recorded messages in a message flow" on page 1908
- "Integration testing overview" on page 1893

About this task

The **ibmint generate tests** command creates the `build.gradle` and `settings.gradle` files required to compile the test project, so that it can be packaged into a BAR file or copied directly into the `run` directory of an integration server, ready for the tests to be run.

The Gradle application is not supplied with App Connect Enterprise, but you can install it from https://gradle.org/install/.

Procedure

1. At the command prompt, change to the directory that contains your test project and then start Gradle, as shown in the following example:

```
# cd /tmp/MyIntegrationTestProject
# gradle
Starting a Gradle Daemon (subsequent builds will be faster)
BUILD SUCCESSFUL in 15s
2 actionable tasks: 2 executed
```

When the command finishes, the tests in the integration test project (in this example, `MyIntegrationTestProject`) are ready to be run.

2. Optional: Build a BAR file containing the test project by using the `ibmint package` command, as shown in the following example:

```
ibmint package --input-path /tmp/MyIntegrationTestProject --output-bar-file /tmp/
MyIntegrationTestProject.bar --do-not-compile-java
```

3. Optional: Run the tests by copying the output directory (which has the same name as the test project) to the `run` directory of your integration server. The application that contains the message

flows required by the test project must also be deployed; if it has not yet been deployed, copy the application BAR file into the integration server's `run` directory.

a) Create an integration server work directory if you have not done so already.
For example:

```
# mqsicreateworkdir /tmp/work-dir
mqsicreateworkdir: Copying sample server.config.yaml to work directory
1 file(s) copied.
Successful command completion.
```

b) Copy the compiled test project to the run directory:

```
# cp -r /tmp/MyIntegrationTestProject /tmp/work-dir/run
```

c) Copy the application BAR file to the integration server run directory, if it is not already deployed:

```
# cp ../MyApplication.bar /tmp/work-dir/run
```

d) Run the **IntegrationServer** command to start the integration server, specifying the name of the test project by using the **--test-project** parameter, as shown in the following example:

```
# IntegrationServer –work-dir /tmp/work-dir --test-project MyIntegrationTestProject
--start-msgflows false
.......2021-01-25 12:16:16.457748: [Thread 44425] (Msg 1/1) BIP9901I: Messages are
now being recorded and stored at '/tmp/recorded_messages'.
.............2021-01-25 12:16:20.274600: [Thread 44425] (Msg 1/1) BIP2155I: About to
'Initialize' the deployed resource 'App1' of type 'Application'.
..2021-01-25 12:16:21.016478: [Thread 44501] (Msg 1/1) BIP2866I: IBM App Connect
Enterprise administration security is inactive.
2021-01-25 12:16:21.040458: [Thread 44501] (Msg 1/1) BIP3132I: The HTTP Listener
has started listening on port '7600' for 'RestAdmin http' connections.
2021-01-25 12:16:21.459 1 STARTING
TEST:App1_Flow1_Pass_through_TestCase_001()
2021-01-25 12:16:21.528 1 FINISHED
TEST:App1_Flow1_Pass_through_TestCase_001() SUCCESSFUL
2021-01-25 12:16:21.542 1
TEST RESULTS:
 com.ibm.mytests.MyIntegrationTestClass:
 App1_Flow1_Pass_through_TestCase_001():SUCCESSFUL
TOTALS:
 PASSED :1
 FAILED :0
 ABORTED :0
 TIME(secs):0.157
```

In this example, the `--start-msgflows false` parameter prevents the deployed message flows from starting and accepting requests through their input nodes, but still allows the tests to run.

Importing a test project into the IBM App Connect Enterprise Toolkit for editing

When you have generated tests for your message flow nodes, you can import them into a test project in the IBM App Connect Enterprise Toolkit, where you can extend them with additional checks. You can also add further tests to the test project.

Before you begin

Read the following topics:

- “Generating tests from recorded messages in a message flow” on page 1908
- “Integration testing overview” on page 1893

About this task

The test project is associated with the specific application that you want to test. The following instructions assume that your test project is imported into the workspace that contains the application to be tested.

Procedure

Import the test project into the IBM App Connect Enterprise Toolkit and then create a BAR file, by completing the following steps:

1. Create an empty test project by right-clicking in the Application Development view and then selecting **New** > **Test Project**.
2. Right-click the new test project, and then select **Import** > **Import generated test files into a test project**.
3. Click **Browse** and navigate to the test project folder (for example, `/tmp/TestProject1`).

   To import the entire test project, select the test project directory to import all the Java source and resource files; when you are prompted to overwrite existing files, select **No to all**. To import only selected test and resource files, use the navigator to select the required files.

   The project references from the imported test files are combined with any existing application references on the test project.
4. Click **Finish** to import the messages into the test project.

Testing a sequence of message flow nodes in a single test case

You can test a sequence of message flow nodes by using the `evaluate()` method or the `propagate()`method.

Before you begin

Read the following topics:

- "Developing integration tests" on page 1891
- "Integration testing overview" on page 1893

About this task

You can test a sequence of message flow nodes, by using either the `evaluate()` method or the `propagate()` method, as described in the following topics:

- "Testing a sequence of nodes by using the evaluate() method " on page 1913
- "Testing a sequence of nodes by using the propagate() method" on page 1914

.

When you use the `propagate()` or `evaluate()` method to test a section of a message flow, you can configure an additional NodeSpy or NodeStub on the message flow, to perform the following actions:

- Observe and validate the message content at any node in the message flow
- Prevent message propagation past a specific message flow node terminal, by using the `setStopAtTerminal()` method
- Substitute the operation of a message flow node

For information about using node stubs on the message flow, see "Replacing a message flow node with a NodeStub in a unit test" on page 1917.

When you are testing a part of a message flow, you can stop the message flow before it completes, by using the `setStopAtInputTerminal()` and `setStopAtOutputTerminal()` methods on a node spy. For information about how to do this, see "Stopping a flow before it completes" on page 1915.

For information about verifying exceptions that were caught while propagating, see "Verifying exceptions" on page 1916.

You can also test an individual message flow node (without propagating the message to any subsequent message flow nodes), by using the `evaluate()` method on a node spy and setting the **`singleNodeTest`** parameter to `true`:

```
 /**
  * Calls the specified input terminal of the message flow node with the supplied message
assembly.
  *
  * @param testAssembly   The message assembly to call the node that contains
  *                       the test time message trees
  * @param singleNodeTest when true, prevents message propagation from any of the
  *                       output terminals on the message flow node
  * @param inputTerminalName The name of the input terminal to call
  * @throws TestException Why the invocation failed
  */
 public void evaluate(final TestMessageAssembly testAssembly, boolean singleNodeTest, String
inputTerminalName) throws TestException
```

Testing a sequence of nodes by using the evaluate() method

You can test a sequence of message flow nodes by using the `evaluate()` method.

Before you begin

Read the following topics:

- "Developing integration tests" on page 1891
- "Testing a sequence of message flow nodes in a single test case" on page 1912

About this task

You can test a sequence of message flow nodes by using the `evaluate()` method and setting the **`singleNodeTest`** parameter to `false`. When the **`singleNodeTest`** parameter is set to `false`, messages can be propagated from any of the output terminals on the node spy. The `evaluate()` method injects the message to the input terminal of a message flow node, and the flow continues either until the injected message reaches the end of the flow, or until an uncaught message-flow exception occurs.

When the `evaluate()` method is called with the **`singleNodeTest`** parameter set to `false`, the sequence of message flow nodes is executed synchronously on the unit test thread. The `evaluate()` method completes when evaluation of the sequence of nodes completes. If any of the evaluated nodes makes a request to other flows (such as a Callable Invoke node), invokes external APIs, or accesses external resources, these must be running and available in order for the `evaluate()` to complete successfully.

In addition to the **`singleNodeTest`** parameter, you can set two other properties on the `evaluate()` method:

- The payload to be propagated.
- Optionally, the name of the terminal to send the payload to. The terminal name (if specified) must be the name of an input terminal. If it is not specified, the 'in' terminal is used.

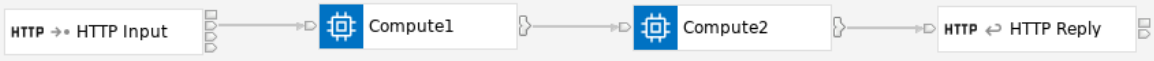


Figure 19. Example message flow

This example calls a message flow node by sending a message to the 'in' terminal of the Compute1 node. The HTTPReply node detects that there was no initial HTTP Input node in scope, so it does nothing. Other transport types might need extra code to prevent errors from being generated.

```
// Define the Spy on the Compute1 that is used to inject the message
SpyObjectReference compute1Reference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("Compute1");
NodeSpy compute1NodeSpy = new NodeSpy(compute1Reference);

// Define the Spy on the Compute2 that is used to verify the required nodes were executed
SpyObjectReference compute2Reference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("Compute2");
NodeSpy compute2NodeSpy = new NodeSpy(compute2Reference);

// Define the Spy on the HTTP Reply that is used to verify the operation
SpyObjectReference httpReplyReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("HTTP Reply");
NodeSpy httpReplyNodeSpy = new NodeSpy(httpReplyReference);

// Declare a new TestMessageAssembly object for the message being sent into the node
TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();

/*
 * Loading of message assembly would be done here
 */

// Send the message assembly to the 'in' terminal of the Compute1 node
compute1NodeSpy.evaluate(inputMessageAssembly, false, "in"); // Note: singleNodeTest = false

// Assert the Compute1 node and Compute2 nodes are both called
assertThat(compute1NodeSpy, nodeCallCountIs(1));
assertThat(compute2NodeSpy, nodeCallCountIs(1));

// Assert the HTTP Reply node is also called
assertThat(httpReplyNodeSpy, nodeCallCountIs(1));
```

When you use the `evaluate()` method to test a section of a message flow, you can configure additional node spies and node stubs on the message flow to perform any of the following actions:

- Observe and validate the message content at any node in the message flow
- Prevent message propagation past a specific message flow node terminal, by using the `setStopAtTerminal()` method
- Substitute the operation of a message flow node

Testing a sequence of nodes by using the propagate() method

You can test a sequence of message flow nodes by using the `propagate()` method.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Testing a sequence of message flow nodes in a single test case” on page 1912

About this task

You can test a sequence of message flow nodes by using the `propagate()` method, which allows the message to be injected to an output terminal of a message flow node. When you use the `propagate()` method, the message flow node itself is not called, but the message is propagated down the flow from the node's output terminal.

When the `propagate()` method is called, the sequence of nodes is executed synchronously on the unit test thread. The `propagate()` method completes when evaluation of the sequence of nodes completes. If any of the evaluated nodes make requests to other flows (such as using a Callable Invoke node), invokes external APIs, or accesses external resources, these must be running and available in order for the propagate to complete successfully.

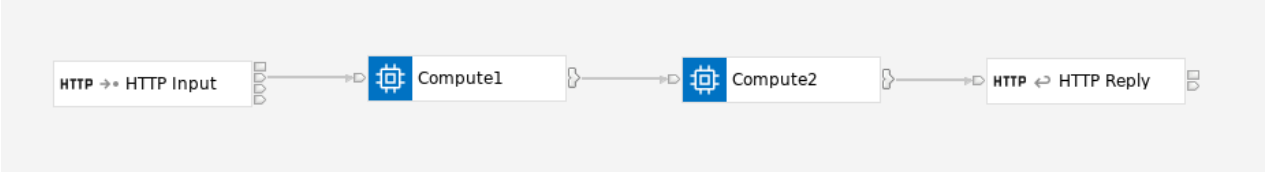


Figure 20. Example message flow

This example propagates a message to the 'out' terminal of the HTTPInput node. The HTTPReply node detects that there was no initial HTTPInput node, so it does nothing. Other transports might need extra code to prevent errors from being generated.

```
// Define the Spy on the HTTP Input which is used to inject the message
SpyObjectReference httpInputReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("HTTP Input");
NodeSpy httpInputNodeSpy = new NodeSpy(httpInputReference);

// Define the Spy on the Compute2 which is used to verify the required nodes were executed
SpyObjectReference compute2Reference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("Compute2");
NodeSpy compute2NodeSpy = new NodeSpy(compute2Reference);

// Define the Spy on the HTTP Reply which is used to verify the operation
SpyObjectReference httpReplyReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("HTTP Reply");
NodeSpy httpReplyNodeSpy = new NodeSpy(httpReplyReference);

// Declare a new TestMessageAssembly object for the message being sent into the node
TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();

/*
 * Loading of message assembly would be done here
 */

// Send the message assembly to the 'out' terminal of the HTTP Input node
httpInputNodeSpy.propagate(inputMessageAssembly, "out");

// Assert the HttpInput node was not called, because we injected the message at the
// output terminal using propagate
assertThat(httpInputNodeSpy, nodeCallCountIs(0));

// Assert the Compute2 node is called
assertThat(compute2NodeSpy, nodeCallCountIs(1));

// Assert the HTTP Reply node is called
assertThat(httpReplyNodeSpy, nodeCallCountIs(1));
```

When you use the `propagate()` method to test a section of a message flow, you can configure additional node spies and node stubs on the message flow to perform any of the following actions:

- Observe and validate the message content at any node in the message flow
- Prevent message propagation past a specific message flow node terminal, by using the `setStopAtTerminal()` method
- Substitute the operation of a message flow node

Stopping a flow before it completes

When you are testing part of a message flow, you can stop the flow before it completes, by using the `setStopAtInputTerminal()` and `setStopAtOutputTerminal()` methods on a node spy.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Testing a sequence of message flow nodes in a single test case” on page 1912

About this task

You can use the `setStopAtInputTerminal()` method on a node spy to stop the execution of a message flow at an input terminal, as if the input terminal was not connected. As a result, the node and any subsequent nodes that it is connected to are not evaluated.

Alternatively, you can use the `setStopAtOutputTerminal()` method to stop the execution of the message flow at an output terminal, as if the output terminal was not connected. As a result, the node is evaluated but subsequent nodes are not.

The following example tests a partial message flow containing MQInput, Compute, and MQOutput nodes. In this example, the `setStopAtInputTerminal()` method stops propagation at the input terminal on the MQOutput node:

```
// Define the Spy on the MQ Input which is used to inject the message
SpyObjectReference mqInputReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example4").node("MQ Input");
NodeSpy mqInputNodeSpy = new NodeSpy(mqInputReference);

// Define the Spy on the Compute node which is used to validate the operation
SpyObjectReference computeReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example4").node("Compute");
NodeSpy computeNodeSpy = new NodeSpy(computeReference);

// Define the Spy on the MQ Output node as we do not want this node to execute
SpyObjectReference mqOutputReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example4").node("MQ Output");
NodeSpy mqOutputNodeSpy = new NodeSpy(mqOutputReference);
mqOutputNodeSpy.setStopAtInputTerminal("in");

// Declare a new TestMessageAssembly object for the message being sent into the node
TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();
/* Loading of message assembly would be done here */

// Declare a new TestMessageAssembly object for the expected message propagated from Compute2
 node
TestMessageAssembly afterComputeExpectedMessageAssembly = new TestMessageAssembly();
/* Loading of message assembly would be done here */

// Send the message assembly to the 'out' terminal of the HTTP Input node
mqInputNodeSpy.propagate(inputMessageAssembly, "out");

// Assert the Compute node is called and that the expected message is returned
assertThat(computeNodeSpy, nodeCallCountIs(1));
assertThat(computeNodeSpy, terminalPropagateCountIs("out", 1));

// Proof that the MQ Output node is not called
assertThat(mqOutputNodeSpy, nodeCallCountIs(0));

// Assert that the actual message assembly matches the expected message assembly
assertThat(actualMessageAssembly, equalsMessage(afterComputeExpectedMessageAssembly));
```

You could achieve a similar result by using the `setStopAtOutputTerminal()` method to stop propagation at the 'out' terminal of the Compute node.

Verifying exceptions

You can verify the details of an exception that was caught during propagation, by inspecting the generated test exception.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Testing a sequence of message flow nodes in a single test case” on page 1912

About this task

If a node in the sequence throws an exception that is not caught, it is returned to the caller of the `propagate()` or `evaluate()` method. Details of a root cause exception can be extracted from the exceptions nested in the **`TestException`**. If the flow catches an exception by using either a TryCatch node or a Catch terminal on a node in the sequence, the exception is processed as usual by the flow and is not returned as a test exception to the `propagate()` method.

The following example shows how the details of an exception generated by an MQGet node (when a queue does not exist) can be verified by inspecting the generated test exception:

```
// Define the Spy on the HTTP Input which is used to inject the message
SpyObjectReference httpInputReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("HTTP Input");
NodeSpy httpInputNodeSpy = new NodeSpy(httpInputReference);

// Define the Spy on the HTTP Reply which is used to verify the operation
SpyObjectReference httpReplyReference = new SpyObjectReference().application("ExamplesApp").
    messageFlow("Example1").node("HTTP Reply");
NodeSpy httpReplyNodeSpy = new NodeSpy(httpReplyReference);

// Declare a new TestMessageAssembly object for the message being sent into the node
TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();

/*
 * Loading of message assembly would be done here
 */

// We expect that the failure thrown by the MQGet node is returned
TestException propagateEx = assertThrows(TestException.class, () -> {
    // Now call propagate on the "out" terminal of the HTTP Input node
    // Note: this approach works equally well with the evaluate method on an input terminal
    httpInputNodeSpy.propagate(inputMessageAssembly, "out");
});

// Verify that one exception in the exception chain has message number BIP266 and contains
// the queue name 'QUEUE1' and the reason 'Unable to open queue'
assertThat(propagateEx, causedBy(hasMessageNumber(2666),
                                 containsMessageText("Unable to open queue"),
                                 containsMessageText("QUEUE1")));
```

Replacing a message flow node with a NodeStub in a unit test

When you are testing a message flow that accesses external resources, you can use a NodeStub in place of calling the external resource.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Testing a sequence of message flow nodes in a single test case” on page 1912

About this task

You can test a message flow without accessing external resources by using a NodeStub trained to return a substitute message assembly instead of accessing the external resources. For example, your message flow might contain a subflow that carries out a database query before returning data back to the main flow. You can create a test case that propagates a message through the main flow, but stubs the behavior of the subflow node by using a NodeStub, so that the test case can be run on a system that does not have access to the database.

For example, a message flow accepts an HTTP input request message and then uses it to look up the price of an item by issuing a database query, before sending back the price details in the reply message:

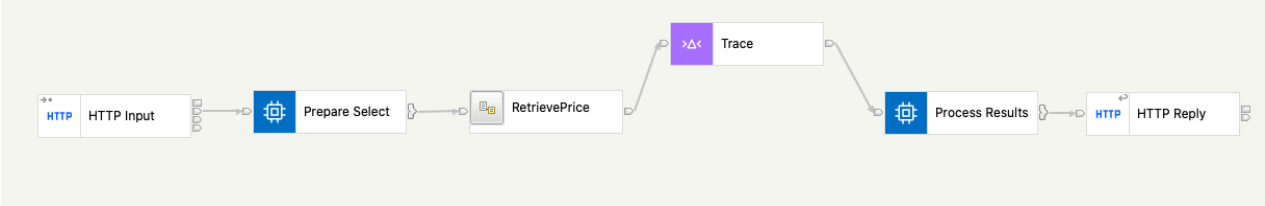


Figure 21. Message flow containing RetrievePrice Subflow

Procedure

Complete the following steps to develop a test case that propagates a message through the main message flow, but uses a NodeStub in place of the RetrievePrice subflow:

1. Define the message assembly that you would expect to be returned by the subflow after it successfully retrieves data from the database.

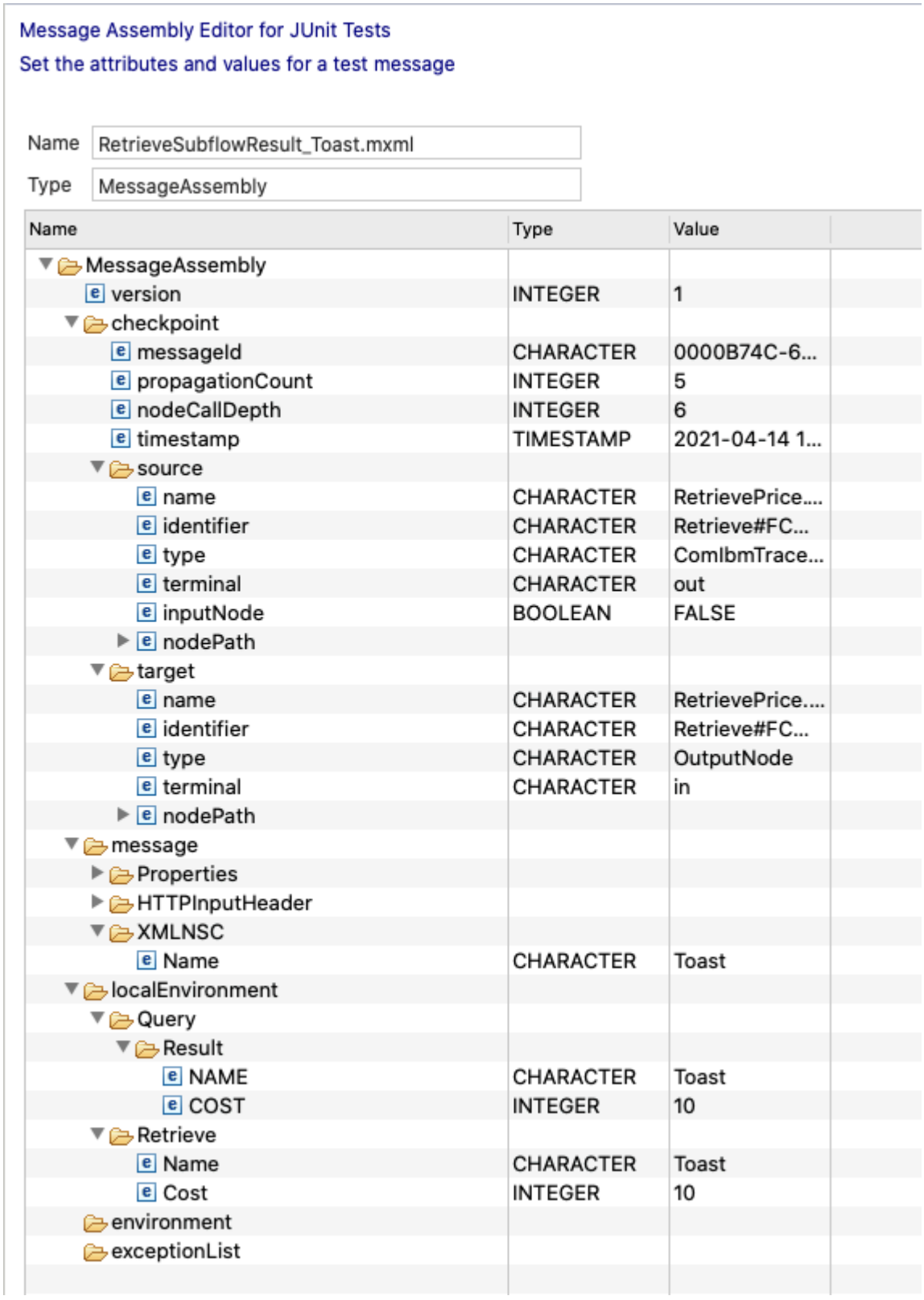


Figure 22. Retrieve subflow result message assembly

You can create the message assembly by recording the message, as described in "Configuring an integration server to record messages" on page 1908.

2. Use the `com.ibm.integration.test.v1.NodeStub` to replace the subflow node, specifying the **SpyObjectReference** object:

```
// Define the Stub on the Subflow, and configure it to return its recorded message
 SpyObjectReference subflowReference = new
 SpyObjectReference().application("AppTwo").messageFlow("Retrieve").node("RetrievePrice");
 NodeStub subflowStub = new NodeStub(subflowReference);
```

3. Use the recorded message assembly to construct a **TestMessageAssembly** object:

```
// Create a Message Assembly and load it with the recorded result for 'Toast'
 TestMessageAssembly subflowResultMessageAssembly = new TestMessageAssembly();
 try {
  String messageAssemblyPath = "/RetrieveSubflowResult_Toast.mxml";
  InputStream messageStream = Thread.currentThread().getContextClassLoader()
    .getResourceAsStream(messageAssemblyPath);
  if (messageStream == null) {
   throw new TestException("Unable to locate message assembly file: " + messageAssemblyPath);
  }
  subflowResultMessageAssembly.buildFromRecordedMessageAssembly(messageStream);
 } catch (Exception ex) {
  fail("Failed to load input message: " + ex.getMessage());
 }
```

4. Use the `com.ibm.integration.test.v1.TrainedBehaviour.propagatesMessage()` method to return the prerecorded message assembly instead of invoking the subflow:

```
// And program stub to return this dummy result instead of calling the subflow
subflowStub.onCall().propagatesMessage("Input", "Output", subflowResultMessageAssembly);
```

   The `propagatesMessage()` method is part of the `TrainedBehaviour` class. You can use this class to create an object that represents the action to be performed by a NodeStub when a message flow node is called.

5. Run the test case by propagating a message to the 'out' terminal of the HTTPInput node, using the propagate method for the NodeSpy on the HTTPInput node:

```
// Now call the flow by propagating the message from the HTTPInput out terminal
httpInputSpy.propagate(inputMessageAssembly, "out");
```

Results

The following complete method code propagates a message through a message flow and uses a NodeStub in place of the `RetrievePrice` subflow:

```
      @Test
public void AppTwo_Retrieve_RetrievePrice_TestCase_001() throws TestException {

 // Define the Spy on the HttpInput node we will use for calling tye flow
 SpyObjectReference httpInputReference = new
SpyObjectReference().application("AppTwo").messageFlow("Retrieve")
   .node("HTTP Input");
 NodeSpy httpInputSpy = new NodeSpy(httpInputReference);

 // Define the Spy on the Process Results compute node we will use for verifying the result
 SpyObjectReference resultsReference = new
SpyObjectReference().application("AppTwo").messageFlow("Retrieve")
   .node("Process Results");
 NodeSpy resultsSpy = new NodeSpy(resultsReference);

 // Declare a new TestMessageAssembly object for the message being sent into the node
 TestMessageAssembly inputMessageAssembly = new TestMessageAssembly();
 try {
  String messageAssemblyPath = "/HttpInput_Toast.mxml";
  InputStream messageStream = Thread.currentThread().getContextClassLoader()
    .getResourceAsStream(messageAssemblyPath);
  if (messageStream == null) {
   throw new TestException("Unable to locate message assembly file: " + messageAssemblyPath);
```

```
   }
   inputMessageAssembly.buildFromRecordedMessageAssembly(messageStream);
  } catch (Exception ex) {
   fail("Failed to load input message: " + ex.getMessage());
  }

  // Create a Message Assembly from the expected output mxml resource
  TestMessageAssembly expectedMessageAssembly = new TestMessageAssembly();
  try {
   String messageAssemblyPath = "/HttpReply_Toast.mxml";
   InputStream messageStream = Thread.currentThread().getContextClassLoader()
     .getResourceAsStream(messageAssemblyPath);
   if (messageStream == null) {
    throw new TestException("Unable to locate message assembly file: " + messageAssemblyPath);
   }
   expectedMessageAssembly.buildFromRecordedMessageAssembly(messageStream);
  } catch (Exception ex) {
   fail("Failed to load input message: " + ex.getMessage());
  }

  // Define the Stub on the Subflow, and configure it to return it's recorded message
  SpyObjectReference subflowReference = new
 SpyObjectReference().application("AppTwo").messageFlow("Retrieve").node("RetrievePrice");
  NodeStub subflowStub = new NodeStub(subflowReference);

  // Create a Message Assembly and load it with the recorded result for 'Toast'
  TestMessageAssembly subflowResultMessageAssembly = new TestMessageAssembly();
  try {
   String messageAssemblyPath = "/RetrieveSubflowResult_Toast.mxml";
   InputStream messageStream = Thread.currentThread().getContextClassLoader()
     .getResourceAsStream(messageAssemblyPath);
   if (messageStream == null) {
    throw new TestException("Unable to locate message assembly file: " + messageAssemblyPath);
   }
   subflowResultMessageAssembly.buildFromRecordedMessageAssembly(messageStream);
  } catch (Exception ex) {
   fail("Failed to load input message: " + ex.getMessage());
  }

  // And program stub to return this fake result instead of calling the subflow
  subflowStub.onCall().propagatesMessage("Input", "Output", subflowResultMessageAssembly);

  // Now call the flow by propagating the message from the HTTPInput out terminal
  httpInputSpy.propagate(inputMessageAssembly, "out");

  // Declare a new TestMessageAssembly object for the actual propagated message
  TestMessageAssembly actualMessageAssembly = resultsSpy.propagatedMessageAssembly("out", 1);

  // Assert that the actual message tree matches the expected message tree
  assertThat(actualMessageAssembly, equalsMessage(expectedMessageAssembly));

 }
```

Editing a message assembly

You can view, edit, and create message assemblies by using the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- “Integration testing overview” on page 1893
- “Developing integration tests” on page 1891

About this task

A message assembly is an object that is used to represent an instance of a message at a given point in time as it is being processed by a message flow, and the contents of the message assembly are accessed and modified by each message flow node as the message is processed. The message assembly contains four trees:

- Message tree
- Environment tree

- Local environment tree
- Exception list tree

The message headers and payload are under the message tree in the message assembly.

You can save a message assembly from the Flow Exerciser when you click on the wire connection in the 'blue-line' view. Alternatively, you can save all message assemblies by selecting the **Save All** button in the Flow Exerciser toolbar.

When you record messages that pass through a message flow, each recorded message is stored as a `.mxml` file, which is a serialized form of a message assembly. These `.mxml` files can then be used in your integration tests.

When you have created a test case from a recorded message, you can view or edit the generated `.mxml` files by using the Message Assembly Editor in the IBM App Connect Enterprise Toolkit. You can also create message assemblies through a test project in the Toolkit.

Procedure

You can view and edit a recorded message, or create a message assembly, by completing the steps in the following tasks:

- "Viewing recorded messages" on page 1922
- "Editing recorded messages" on page 1923
- "Creating a message assembly" on page 1926

For example message assemblies, see the following topics:

- "Example message assembly (XMLNSC)" on page 1929
- "Example message assembly (JSON)" on page 1931

Viewing recorded messages

You can view recorded messages by using the Flow Exerciser in the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- "Editing a message assembly" on page 1921
- "Developing integration tests" on page 1891

About this task

When the Flow Exerciser is in record mode, a blue line shows the path that a message has taken through a message flow. You can click on any wire connection to see a read-only view of the message assembly between two message flow nodes.

While you are viewing a message assembly, you can save it by clicking the **Save** button. The message assembly file name is generated automatically, but you can choose a different file name. The message assembly is saved in the application that contains the message flow that is being recorded in the Flow Exerciser. You can save message assemblies for all wire connections in the blue-line view, by clicking **Save All**. All message assemblies are then saved in the application.

Procedure

1. Select an input message to test your message flow with the Flow Exerciser, as described in "Testing your message flow by using the Flow Exerciser" on page 648.

   When a message has been sent through the message flow, a blue line indicates the path that was taken through the flow.

2. Click a highlighted connection to view the data that passed through it. A read-only view of the message assembly is displayed.

   If more than one message was sent through the message flow, or if a single message passed through a connection multiple times, you can see the message assembly for each message instance by selecting or clicking through the message numbers at the top of the view.
3. Click **Save** to save the message assembly in the application that contains the message flow.

   You can either accept the file name that is generated automatically, or specify a new file name.

   As an alternative to clicking each connection, you can save the message assembly for every wire connection that is highlighted by clicking **Save All**.

Editing recorded messages

You can edit recorded messages by using the Message Assembly Editor in the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- "Editing a message assembly" on page 1921
- "Developing integration tests" on page 1891

About this task

When you have saved a message assembly file, you can open it in the Message Assembly Editor and make changes to the elements in each of the four trees that it contains. You can edit entries in the **Name**, **Type**, and **Value** columns, and you can add new entries to each of the trees.

You can also add HTTP headers (`HTTPInputHeader`,`HTTPReplyHeader`, `HTTPRequestHeader`, and `HTTPResponseHeader`) and MQ headers (`MQMD`, `Dead Letter`, `MQRFH2`, and `MQRFH2C`). Each header has an initial set of values, which you can modify. You can use the Local Environment wizard to select parts of the local environment to add to your message assembly.

Procedure

1. To open a message assembly (`.mxml`) file in the Message Assembly Editor, either double-click the `.mxml` file or right-click the file and then click **Open with** > **Message Assembly Editor**.

   The message assembly is shown in the Message Assembly Editor.

2. In the Message Assembly Editor, you can open any of the trees in the message assembly and change the entries in the **Name**, **Type**, and **Value** columns.

   You can change an entry in the **Name** or **Value** column by clicking in the cell and modifying the entry that it contains. You can change an entry in the **Type** column by clicking a cell and selecting the required type from the drop-down list.

   Mixed content is not supported in a message assembly file; as a result, if an element has a type and value in the message body and you add a child element, the entries in the **Value** and **Type** columns are removed.

   You can select **Copy ESQL Path** or **Copy XPath** on an element. The path to the element is copied to the clipboard, and you can then paste it into the ESQL Compute Node Editor or the Java Compute Node Editor.

   You can remove whole sections from the message assembly by selecting the element and then clicking **Delete** in the context menu.

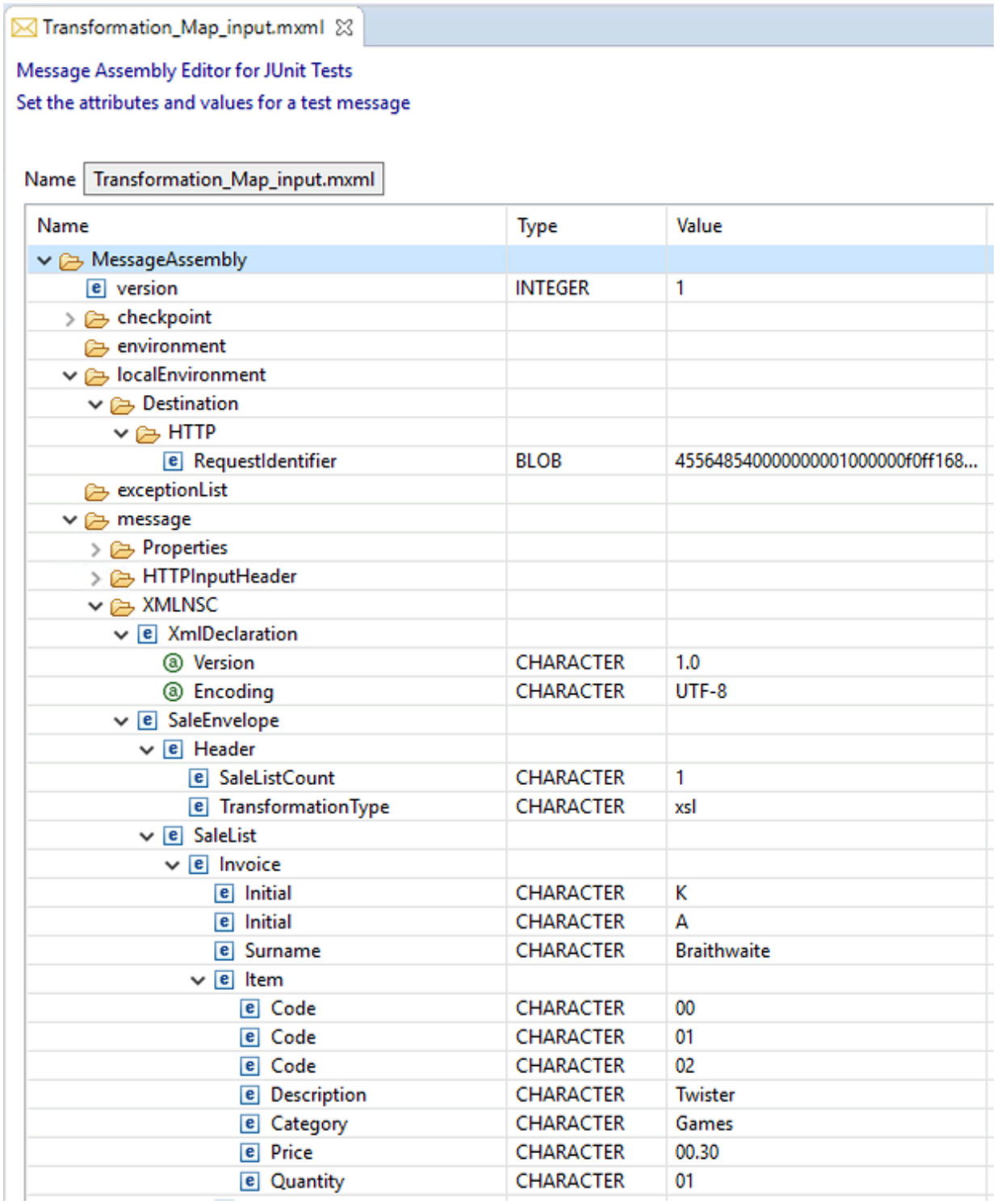


Figure 23. Message Assembly Editor for JUnit tests

3. When you have finished editing the message assembly, click **File** > **Save** to save the file.

   The Message Assembly is saved in the Application that contains the message flow, and the modified `.mxml` file is now available for use in your integration test case.

Creating a message assembly

You can create message assemblies by using the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- "Integration testing overview" on page 1893
- "Developing integration tests" on page 1891

About this task

A message assembly is an object that is used to represent an instance of a message at a given point in time as it is being processed by a message flow. The message assembly contains the message headers and payload (which can be XML, JSON, DFDL or BLOB data), together with the environment tree, the local environment tree, and the exception list tree. The contents of the message assembly are accessed and modified by each message flow node as the message is processed.

For examples of messages that were created by using the Message Assembly Editor, see the following topics:

- "Example message assembly (XMLNSC)" on page 1929
- "Example message assembly (JSON)" on page 1931

Video: Understanding a message assembly

This video demonstrates the capability to view and edit a message assembly.

Procedure

You can create a message assembly by completing the following steps:

1. In the Application Development view of the IBM App Connect Enterprise Toolkit, right-click the test project that will contain your message assembly, and then click **New** > **Message Assembly**.
2. In the **Create a message assembly** window, enter a name for your new message assembly and then click **Finish**.

   The Message Assembly Editor is displayed.

3. Right-click the `MessageAssembly` tag in the displayed table. A menu is displayed, which you can use to create each tree in the message assembly (Environment, Local Environment, Exception List, and Message):

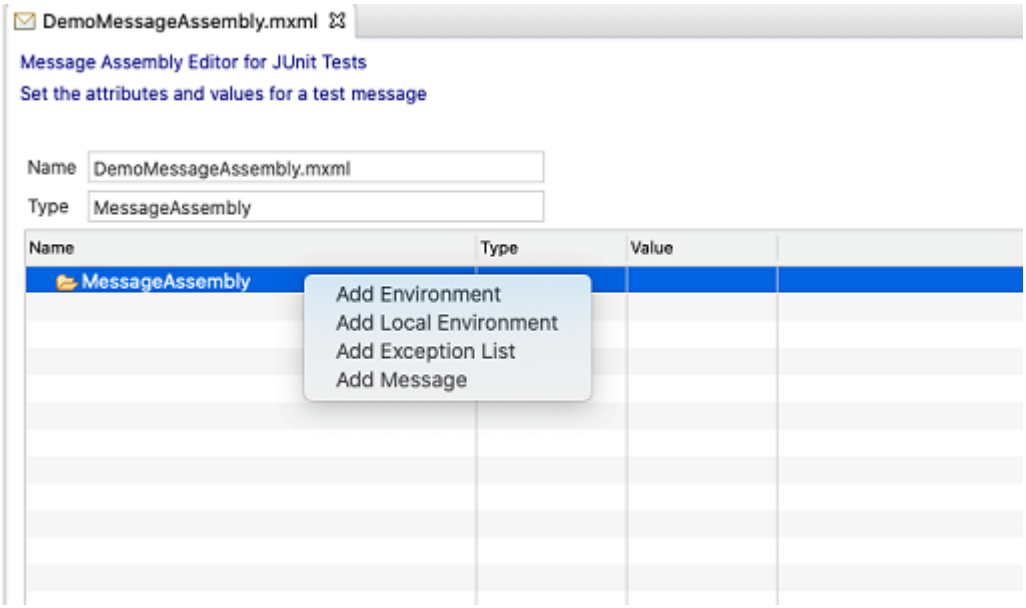


Figure 24. Message Assembly Editor context menu

4. Add an environment tree to the message assembly:
   a) Select **Add Environment** to add an environment tree. An `environment` tag is added to the message assembly.
   b) Add elements to the environment tree, by right-clicking the `environment` tag and then clicking **Add child** > **Element**.
   c) Add attributes to the element by right-clicking the element and then clicking **Add Attribute**.

5. Add the local environment tree to the message assembly:
   a) Right-click the `MessageAssembly` tag and click **Add Local Environment**.
      The **Create Local Environment Tree** window is displayed.
   b) Select the entries that you want to add to the local environment tree.
      You can expand the entries and select individual elements or folders.
   c) When you have finished adding entries to the local environment tree, click **Finish**.
      The local environment entries are shown in the Message Assembly Editor.

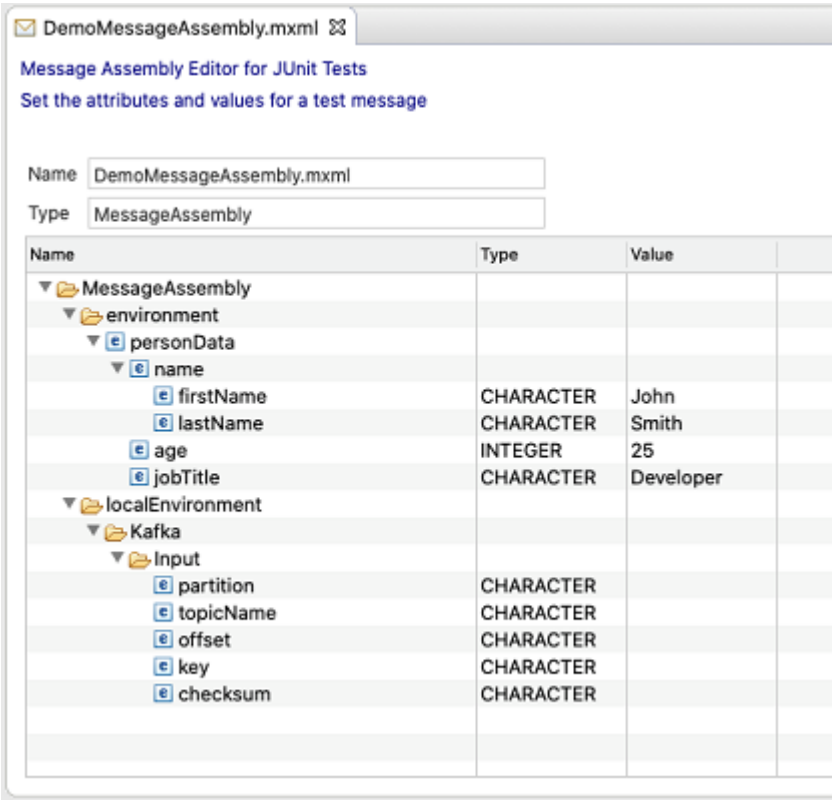


Figure 25. Environment and local environment trees in the Message Assembly Editor

      You can add further entries to the local environment tree by right-clicking the `localEnviroment` element and then clicking **Select and add elements**.
   d) Click **Finish**.
6. Add an exception list tree to the message assembly:
   a) Right-click the `MessageAssembly` tag and then click **Add Exception List**.
   b) To add entries to the exception list tree, right-click `exceptionList`, click **Add Exception List entry**, and then select the specific types of exception that you want to add to the tree.
      The entries are added to the exception list tree. You can also add nested exceptions and extra inserts.

7. Add the message tree to the message assembly:
   a) Right-click the `MessageAssembly` tag and then click **Add Message**.
      The **Create Message Tree** window is displayed.
   b) Select the transport protocol and message body parser for the message tree.
      - Choose one of the following transport protocols: `HTTP`, `SOAP`, `MQ`, `TCPIP`, `File`, or `Other`.

        If you select `HTTP`, an `HTTPInputHeader` is added automatically. If you select `MQ`, an `MQMD` is added automatically.

        The value for `ReplyProtocol` in the Properties tree is set based on the selected transport. For example, if you select `MQ`, the **`ReplyProtocol`** property is set to `MQ`.
      - Choose one of the following message body parsers: `XMLNSC`, `JSON (Object)`, `JSON (Array)`, `DFDL`, or `BLOB`.
      - Set values for the properties and headers that you require for your unit test.

        XMLNSC and DFDL message body parsers require a root element, which you can add by selecting them from the context menu. For XMLNSC, you can add elements and attributes. For DFDL, you can add elements but no attributes.

        For a JSON Object, you can add a child element that is a JSON Object or JSON Array. For a JSON Array, you can add a child element that is a JSON Array Item. For a JSON Array Item, you can add a child element that is a JSON Object or JSON Array.

        Elements that contain child elements do not have a type associated with them, but you can associate a type with each of the child elements. When you select a type, validation is performed and the type is set in the **`iib:valueType`** attribute.
   c) Click **Finish**.
      The properties tree and message domain element are added under the message.
   d) Add message headers by right-clicking the `message` element and then clicking either **Add HTTP Header** or **Add MQ Header**, and then selecting the type of header.
      - For `HTTP`, you can add one of the following headers: `Input`, `Reply`, `Request`, or `Response`.
      - For `MQ`, you can add one of the following headers: `MQMD`, `Dead Letter`, `MQRFH2`, or `MQRFH2C`.
8. Save your message assembly.

Example message assembly (XMLNSC)

You can create message assemblies by using the Message Assembly Editor in the IBM App Connect Enterprise Toolkit.

The following example shows an XMLNSC message that was constructed by using the Message Assembly Editor:

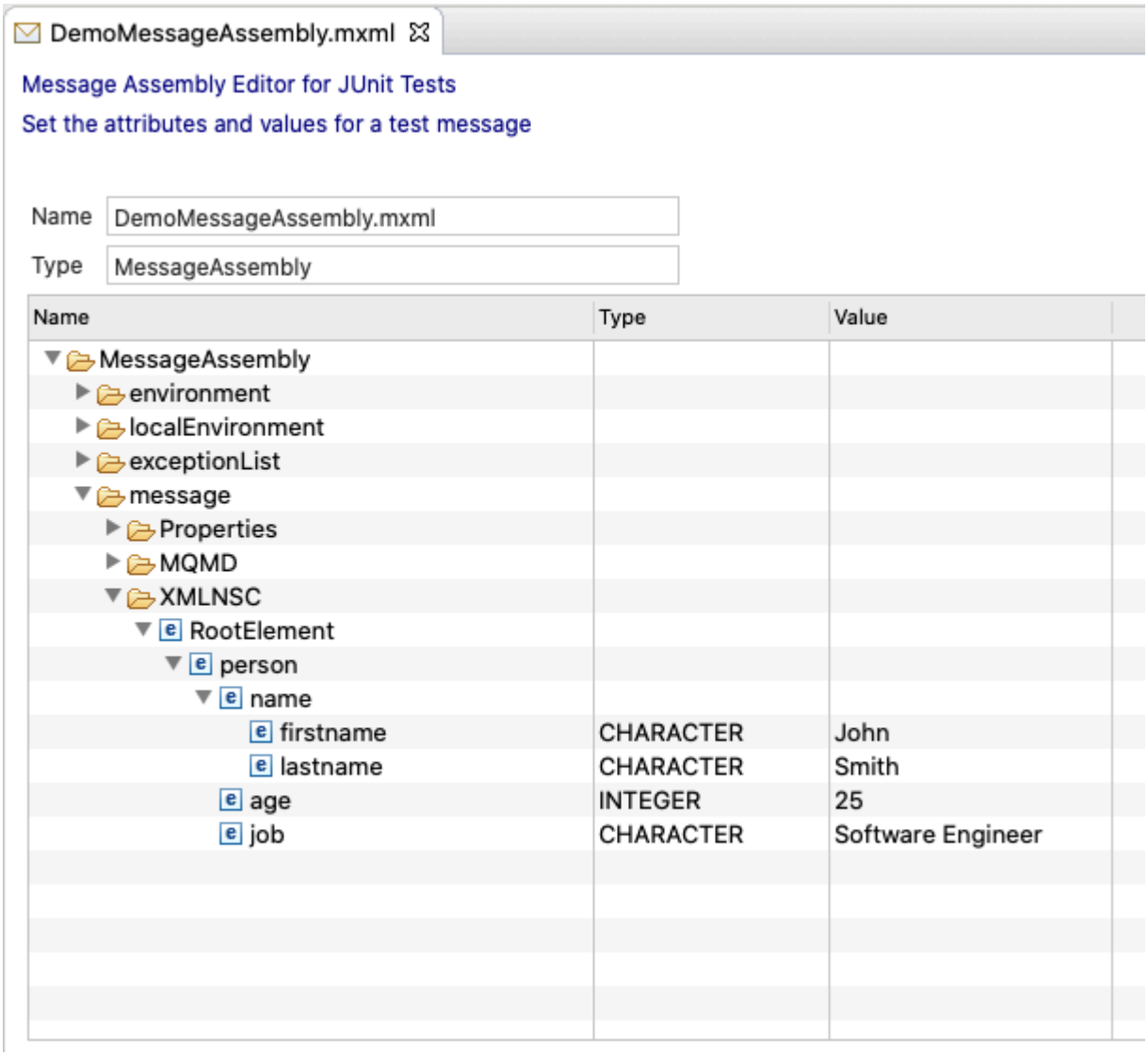


Figure 26. XMLNSC message constructed using the Message Assembly Editor

When this message was created in the Message Assembly Editor, the following entries were saved to the `.mxml` file under the XMLNSC element of the message tree:

```
<XMLNSC iib:parser="xmlnsc">
    <RootElement>
       <person>
          <name>
             <firstname iib:valueType="CHARACTER">John</firstname>
             <lastname iib:valueType="CHARACTER">Smith</lastname>
          </name>
          <age iib:valueType="INTEGER">25</age>
          <job iib:valueType="CHARACTER">Software Engineer</job>
       </person>
    </RootElement>
 </XMLNSC>
```

For information about how to create message assemblies by using the Message Assembly Editor, see "Creating a message assembly" on page 1926. For information about using message assemblies in unit tests, see "Developing integration tests" on page 1891.

Example message assembly (JSON)

You can create message assemblies by using the Message Assembly Editor in the IBM App Connect Enterprise Toolkit.

The following example shows a JSON message that was constructed by using the Message Assembly Editor:

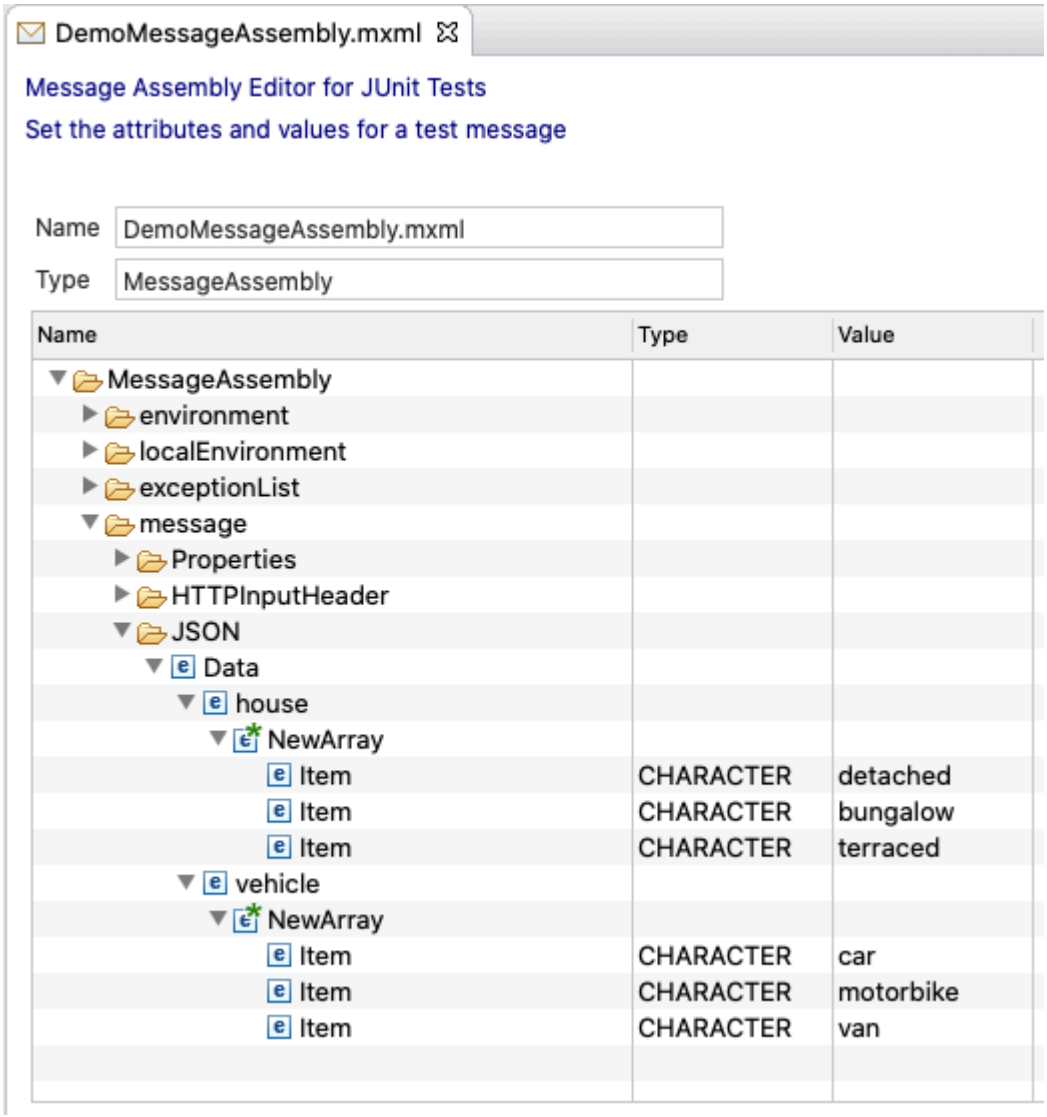


Figure 27. JSON message constructed using the Message Assembly Editor

When this message was created in the Message Assembly Editor, the following entries were saved to the `.mxml` file under the JSON element of the message tree:

```
<JSON iib:parser="json">
   <Data>
      <house>
         <NewArray iib:elementType="0x01001000">
            <Item iib:valueType="CHARACTER">detached</Item>
            <Item iib:valueType="CHARACTER">bungalow</Item>
            <Item iib:valueType="CHARACTER">terraced</Item>
         </NewArray>
      </house>
   <vehicle>
         <NewArray iib:elementType="0x01001000">
            <Item iib:valueType="CHARACTER">car</Item>
            <Item iib:valueType="CHARACTER">motorbike</Item>
            <Item iib:valueType="CHARACTER">van</Item>
         </NewArray>
```

```
            </vehicle>
         </Data>
      </JSON>
```

For information about how to create message assemblies by using the Message Assembly Editor, see “Creating a message assembly” on page 1926. For information about using message assemblies in unit tests, see “Developing integration tests” on page 1891.

Running integration tests

You can run integration tests by using the IBM App Connect Enterprise Toolkit or the command line.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Integration testing overview” on page 1893

About this task

Test projects are the top-level artifacts in App Connect Enterprise that contain your test cases. All the test cases for an application are stored in a test project, with the following structure:

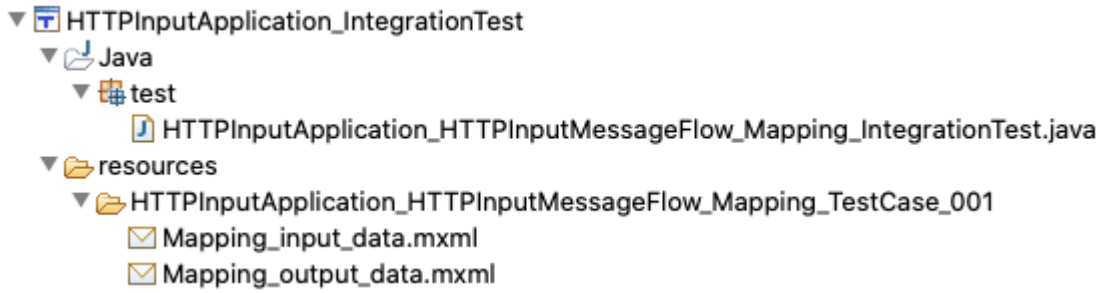


Figure 28. Test project structure

The test cases are methods in a Java Class that can use resources in the test project. The resources are in a folder called `resources`, under the test project. The test project contains a reference to the application that is under test, and you run the test project in order to run the tests that it contains.

Procedure

To run your integration tests, complete the steps in one of the following topics:

- “Running tests in the IBM App Connect Enterprise Toolkit” on page 1933
- “Running tests by using the command line” on page 1932

Optionally, you can configure a specific integration server to run the tests, and you can add an existing test project to a BAR file to deploy and run it, by following the steps in the following topics:

- “Configuring an integration server to run the test case” on page 1935
- “Adding a test project to a BAR file” on page 1936

Running tests by using the command line

You can run tests in a test project by using the command line.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Running integration tests” on page 1932

About this task

You can run tests in a test project by using the **IntegrationServer** command to start the integration server, specifying the name of the integration test project with the **--test-project** parameter. For example:

```
IntegrationServer –work-dir /tmp/work-dir --test-project MyIntegrationTestProject --start-
msgflows false
```

where the **--work-dir** parameter specifies the work directory of the integration server that will run the tests, and the **--test-project** specifies the name of the test project to be run. The *false* value on the **--start-msgflows** parameter prevents the deployed message flows from starting and accepting requests through their input nodes, but still allows the tests to run.

In addition to the parameters used in the example, the following optional flags can be used when running test projects in a pipeline, to speed up test operation:

- Use the **--admin-rest-api -1** parameter to disable the REST admin listener
- Use the **--no-nodejs** parameter to disable Node.js
- Use the **--stop-after-duration <seconds>** parameter to configure a maximum timeout in seconds, after which the integration server will terminate if the tests have not completed

Video: Generate and run tests from the command line

This video demonstrates how to generate and run tests from the command line.

Running tests in the IBM App Connect Enterprise Toolkit

You can run the unit tests from the IBM App Connect Enterprise Toolkit by running the test project.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Running integration tests” on page 1932

About this task

You can run unit tests by running their test project in the IBM App Connect Enterprise Toolkit.

Procedure

1. In the Application Development view of the IBM App Connect Enterprise Toolkit, right-click the test project and then click **Run test project**:

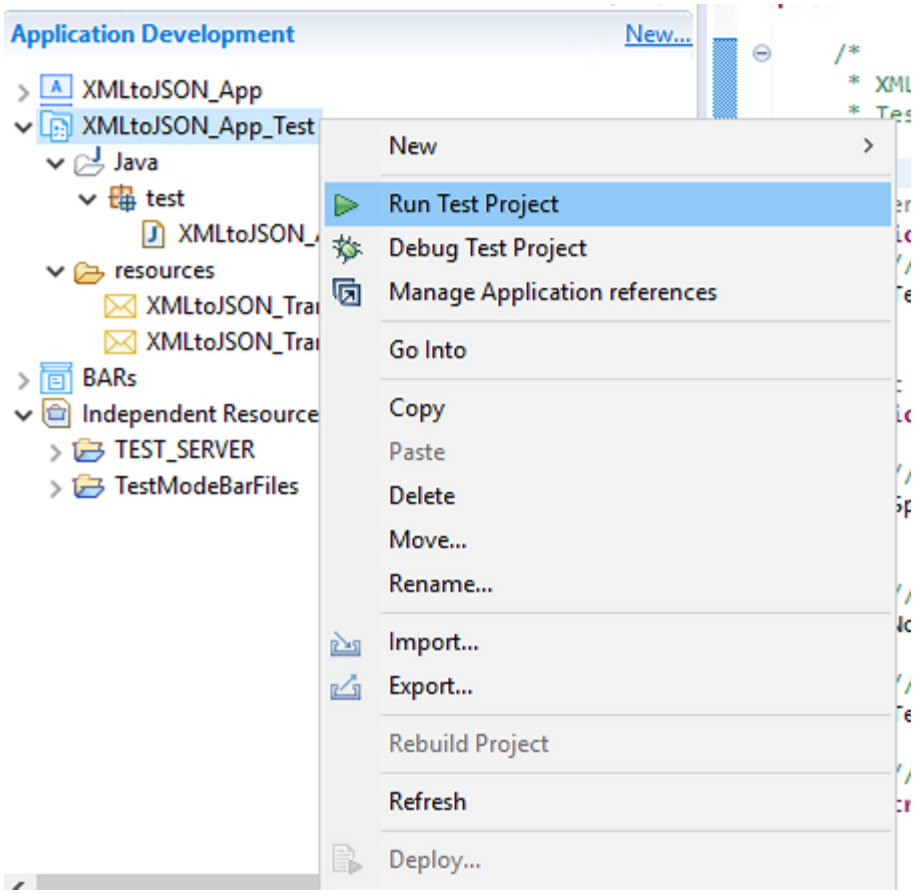


Figure 29. Running a test project in the Toolkit.

2. When the test project is run, a BAR file is generated. This BAR file is automatically deployed to an integration server, which starts with the **`--test-project`** argument. The BAR file automatically includes any applications that are referenced by the test project.

   By default, the tests are run on a temporary integration server, but you can configure an alternative integration server by following the steps in "Configuring an integration server to run the test case" on page 1935.

   The BAR file that is generated is stored in the `GeneratedBarFiles` folder. For example: `/GeneratedBarFiles/MyApplication_IntegrationTestproject.generated.bar`

3. You can view the test results by switching to the JUnit view in the Toolkit. The test results are displayed in the JUnit results panel.

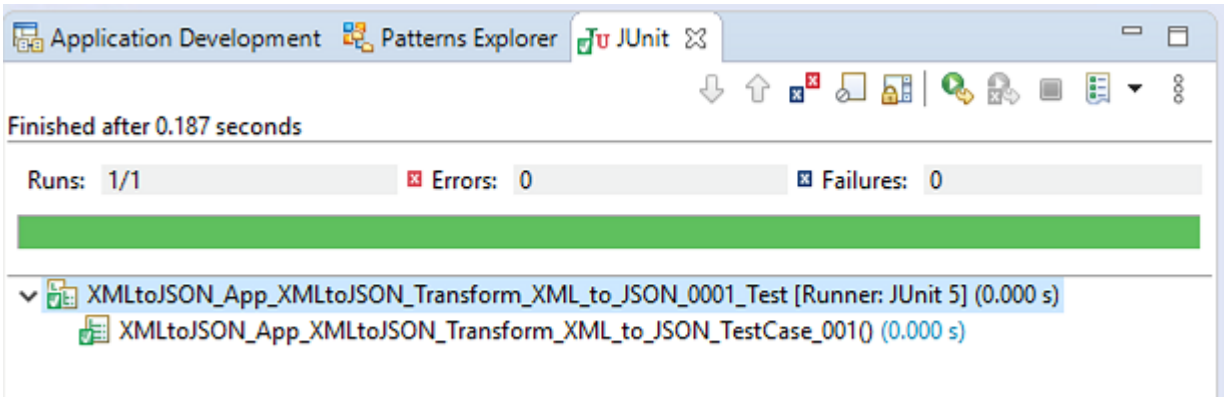


Figure 30. Unit test results in the JUnit view.

4. You can see the output from the integration server console by switching to the Console view.

Figure 31. Integration server output in the Console view.

Configuring an integration server to run the test case

By default, tests are run on a temporary integration server, but you can configure an alternative integration server to run them.

Before you begin

Read the following topics:

- "Developing integration tests" on page 1891
- "Running integration tests" on page 1932

About this task

When you run a test project, a launch configuration is created automatically. You can then change the configuration that is used to run the tests, including the integration server that runs them, by editing this launch configuration.

Procedure

1. To edit the launch configuration, select **Run configurations** from the **Run** menu in the Toolkit.
   The **Create, manage, and run configurations** window is displayed.

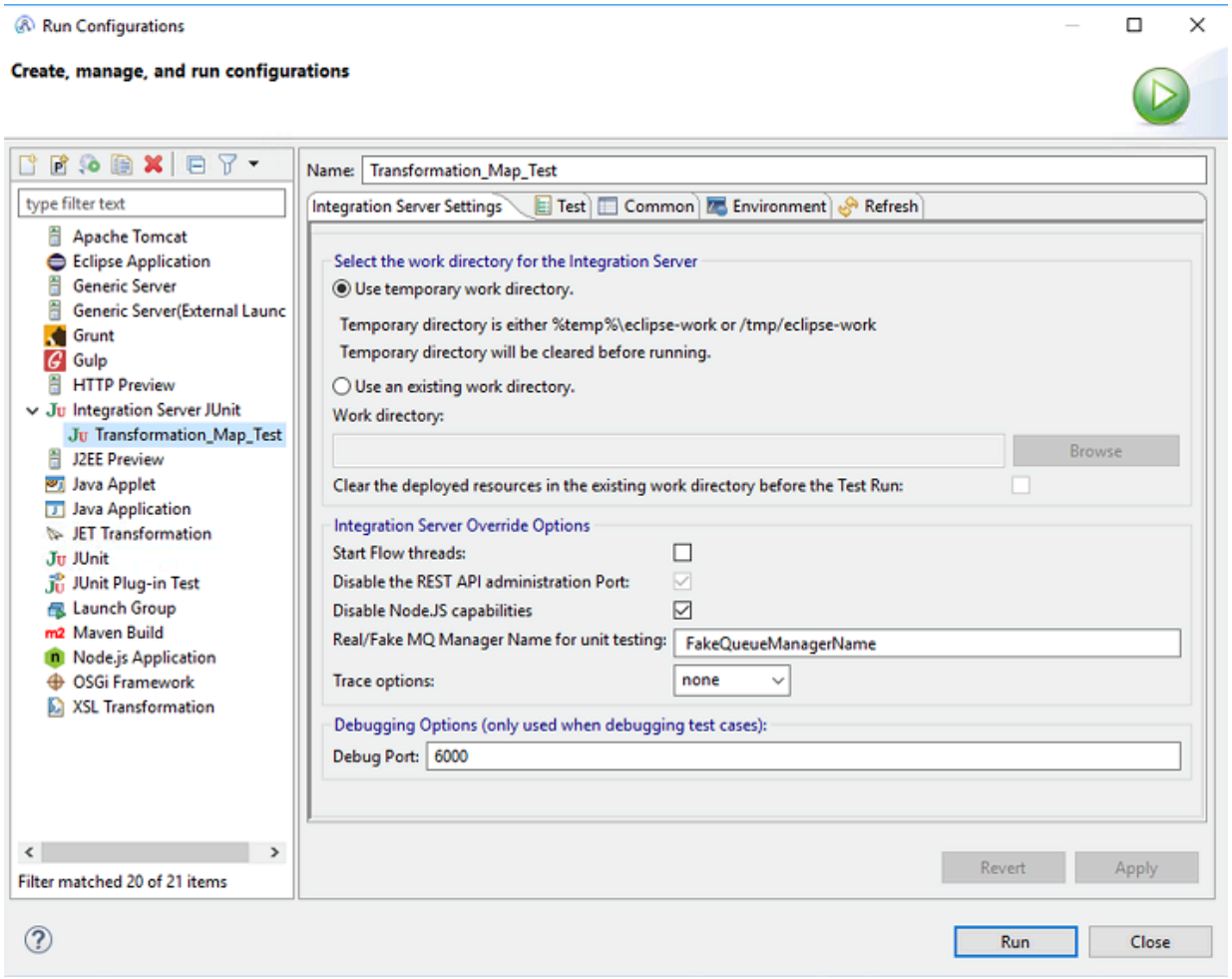


Figure 32. Create, manage, and run configurations window

2. Specify the name of the test project in the **Name** field.
3. Choose an integration server to run the test case, by specifying the location of its work directory in the **Integration Server Settings** tab.

   You can also choose to start flow threads, specify a debug port, and run a service trace or diagnostic trace while running the test case.
4. Click **Run** to launch the tests on the specified integration server.

Adding a test project to a BAR file

You can add a test project to an existing BAR file by using the BAR file editor.

Before you begin

Read the following topics:

- “Developing integration tests” on page 1891
- “Running integration tests” on page 1932

About this task

To add a test project to an existing BAR file, complete the following steps:

Procedure

1. Open the BAR file in the BAR file editor, and then select **Applications, shared libraries, services, REST APIs, and Test Projects**.
2. Select **Test Projects** from the list of deployable resources to include in the BAR file.

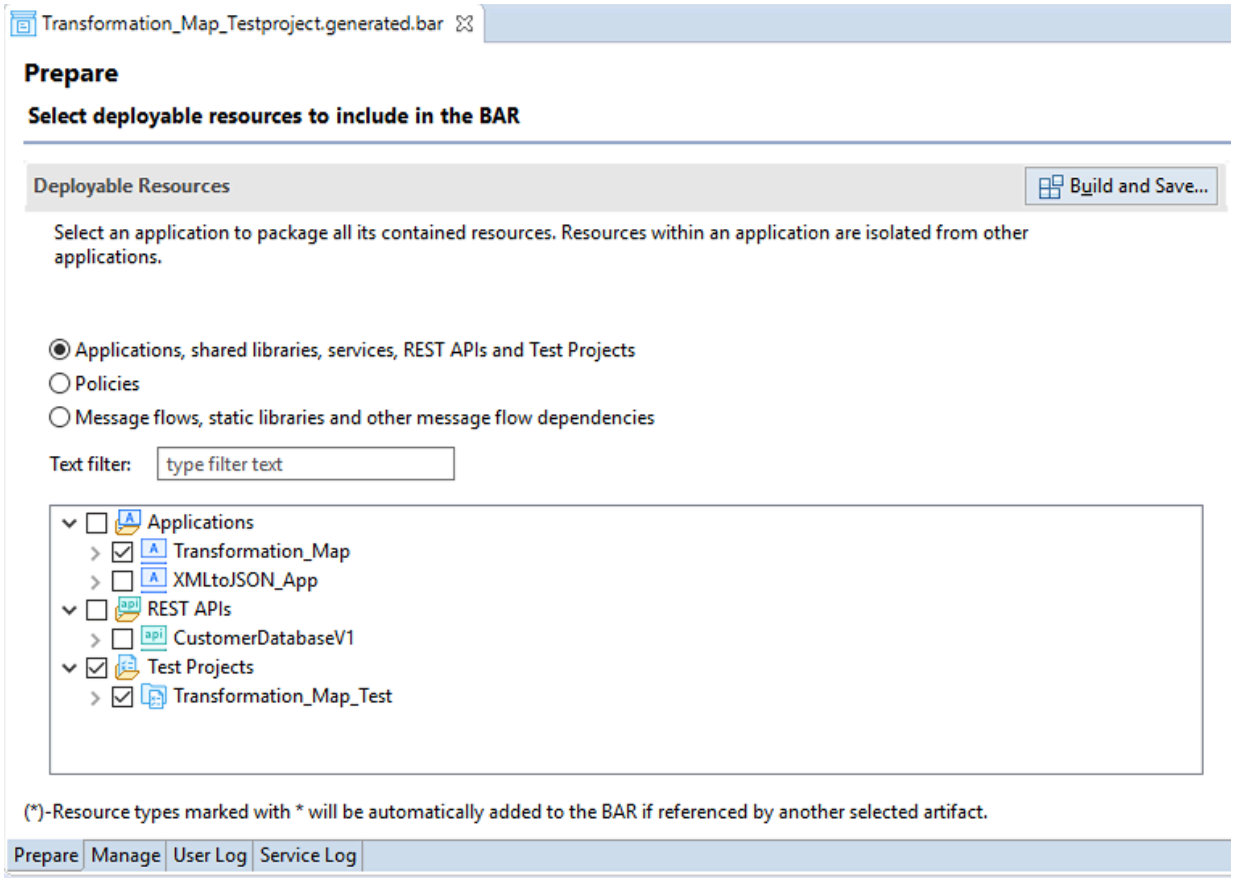


Figure 33. Add test project to BAR file

3. If you are creating a new BAR file that contains a test project, ensure that you also add the application that is being tested, so that the test project and the application can be extracted to the `run` directory of the integration server's work directory.

   The **Manage** tab shows the test project and the application under test:

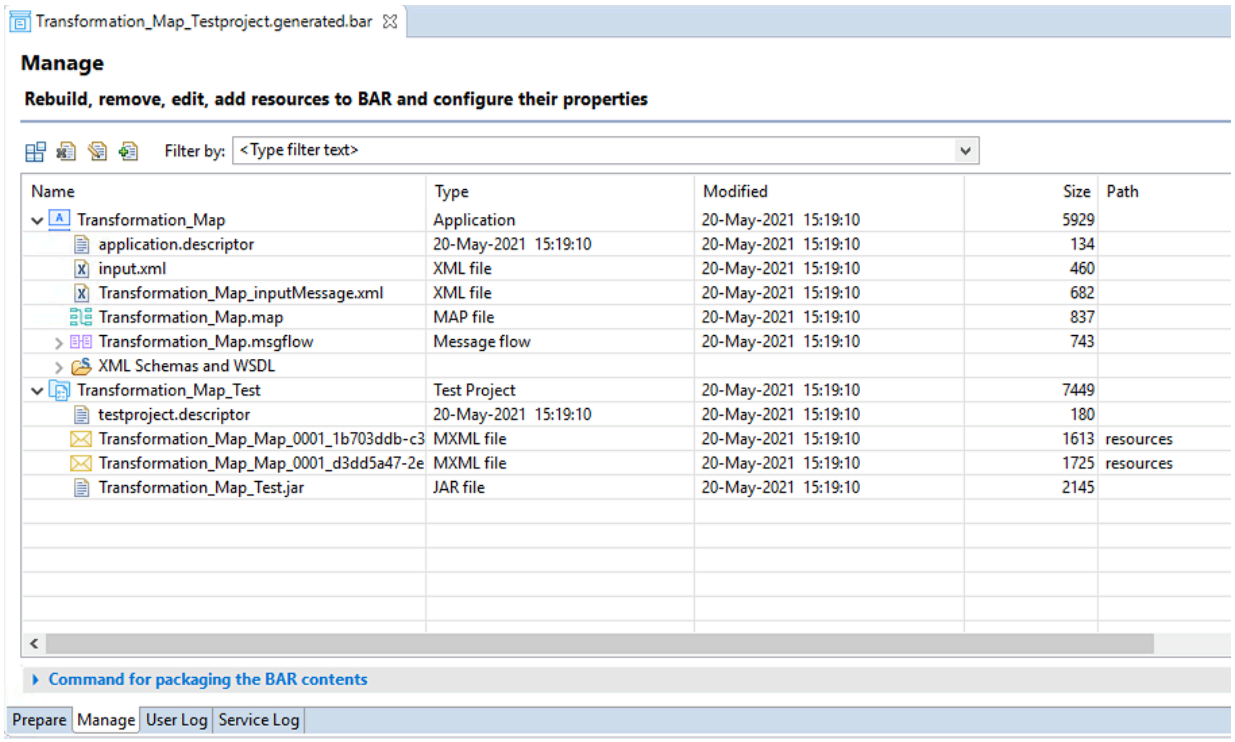


   Figure 34. Manage tab showing the test project and application to be tested

4. When the test project is run, the Java class containing the test case is compiled. The `.jar` file that contains the built classes is added in the BAR file under the test project. The resources that are used by the test cases are also added to the test project.

Example message flow node tests

You can create unit tests for message flow nodes by using the IBM App Connect Enterprise Toolkit.

Before you begin

Read the following topics:

- "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894
- "Integration testing overview" on page 1893

About this task

To test a message flow node, you must create a message assembly with which to invoke the node, and then examine the contents of the message assembly to check whether the operation of the message flow node was successful. For more information about message assemblies, see "Editing a message assembly" on page 1921.

You can create a unit test for a message flow node by following the instructions in "Developing integration tests by using the IBM App Connect Enterprise Toolkit" on page 1894.

Procedure

The following examples show segments of code generated by the Create Test Case wizard:

- **Creating a message assembly from a recorded message**

  The following code segment loads a `.mxml` file from the resource directory associated with the test project. The `.mxml` file contains the message body as well as property headers, the environment tree, the local environment tree, and the exception list tree.

```
// Declare a new TestMessageAssembly object
TestMessageAssembly messageAssembly = new TestMessageAssembly();
try {
 String msgPath = "/HTTP-Input_0000CE13-600ED4AE-00000001-0.mxml";
 InputStream messageStream =

Thread.currentThread().getContextClassLoader().getResourceAsStream(msgPath);
 if (messageStream == null) {
 throw new TestException("Unable to locate message assembly file:"+msgPath);
 }
 messageAssembly.buildFromRecordedMessageAssembly(messageStream);
}
catch (Exception ex) {
 fail("Failed to load input message: " + ex.getMessage());
}
```

- **Creating a message assembly from a JSON string**

  The following code segment generates a message assembly that contains a JSON message tree for the message body, together with a default Properties folder. The environment, local environment and exception list trees are empty.

```
// Declare a new TestMessageAssembly object
String sJSON = “{ \”name\”: { \”first\”: \”John\”, \”last\”: \”Doe\” } }”;
TestMessageAssembly messageAssembly = new TestMessageAssembly();
inputMessageAssembly.buildMessageBodyFromJSON(sJSON);
```

- **Creating a message assembly from an XML string**

  The following code segment generates a message assembly that contains a JSON message tree for the message body, together with a default Properties folder. The environment, local environment and exception list trees are empty.

```
// Declare a new TestMessageAssembly object
String sXML = “<name><first>John</first><last>Doe</last></name>”
TestMessageAssembly messageAssembly = new TestMessageAssembly();
inputMessageAssembly.buildMessageBodyFromXML(sXML);
```

- **Invoking a message flow node by using a message assembly**

  If you have already created a message assembly, either by loading a recorded message or by building it directly, you can invoke a message flow node with the message assembly. The **SpyObjectReference** object is used to build a path to the message flow node to be tested. The application and the message flow that contains the message flow node to be tested must be deployed to the integration server alongside the test project.

```
// Locate the message flow node to be tested
SpyObjectReference nodeReference = new SpyObjectReference()
 .application("ApplicationOne").
 .messageFlow("FlowOne")
 .node("My Compute Node");
// Create a spy on the message flow node to be tested
NodeSpy nodeSpy = new NodeSpy(nodeReference);
// Invoke the message flow node with the message assembly prepared earlier
nodeSpy.invoke(inputMessageAssembly);
```

- **Extracting the output message assembly from a message flow node**

  After a message flow node has been invoked, in order to verify that it operated correctly you must extract the output message assembly from the node spy.

```
// Assert the terminal propagate count for the message
assertThat(nodeSpy, terminalPropagateCountIs("out", 1));
assertThat(nodeSpy, terminalPropagateCountIs("failure", 1));
// Retrieve the outupt message from the Node spy
TestMessageAssembly actualMessage = nodeSpy.propagatedMessageAssembly("out",
```

```
1);
```

The first parameter is the terminal from which to retrieve the output message, and the second parameter is the message number. When a message flow node is invoked, more than one output message assembly can be generated, which means that your test must check all terminals for the expected number of messages as well as the message content. Note that the terminal names are the names that are defined internally and typically start with a lower-case letter, rather than the terminal names that are displayed in the Toolkit, which typically start with an upper-case letter.

- **Comparing the output message using the App Connect Enterprise Test Message Assembly matcher**

  The App Connect Enterprise Test Message Assembly matcher compares two message assemblies and reports details on the first difference it detects. The most common use of this matcher is to compare against an expected message assembly that has been loaded from a previously recorded message.

```
// Assert that the actual message assembly matches the expected message assembly
assertThat(actualMessageAssembly, equalsMessage(expectedMessageAssembly));
```

  Comparing the entire content of a message can reduce the stability of a test, in which an apparently insignificant change in message content can cause the test to fail. You can use the **`ignoreTimeStamps`** method in the matcher to make a test more robust against expected or insignificant parts of the message tree:

```
// Assert that the actual message assembly matches the expected message assembly
assertThat(actualMessageAssembly,
 equalsMessage(expectedMessageAssembly).ignoreTimeStamps());
```

- **Comparing the output message using alternative Hamcrest matchers**

  In addition to using the JSON Unit and XMLUnit Hamcrest matchers that are provided with App Connect Enterprise, you can use third-party matchers.

```
// Retrieve a JSON string representation of the Message body
String expectedJSON = “{ \”name\”: { \”first\”: \”John\”, \”last\”: \”Doe\” } }”;
String actualJSON = messageAssembly.getMessageBodyAsJSON();
assertThatJson(expectedJSON, jsonEquals(actualJSON));
```

- **Using Test Exception matchers**

  The **`TestException`** class is used to report exceptions in the integration test framework, including the following types of failure:

  – Operational test failures such as trying to locate a message flow node in an application that has not been deployed

  – Execution test failures in which a message flow node is invoked with a message and the message flow node has thrown an exception and neither the failure or catch terminals are attached

  Test exceptions can form a chain, so when a test exception is checked it might be necessary to navigate the chain of exceptions in order to locate the root cause. App Connect Enterprise provides a number of Hamcrest matchers to validate test exceptions.

```
try {
 // Invoke the message flow node with the message assembly prepared earlier
 nodeSpy.invoke(inputMessageAssembly);
}
catch (TestException te) {
 // Verify the top level exception is message number BIP3687
 asserThat(te, hasMessageNumber(3687));
 // Verify the root exception contains the message text “Parser failure”
 assertThat(te.getRootCause(), containsMessageText("Parser failure")));
 // Verify that an exception somewhere in the exception chain contains
 // message number BIP2112 and the message text contains the work ‘Broken’
 assertThat(te, causedBy(hasMessageNumber(2112),
containsMessageText("Broken")));
}
```

Developing integration solutions from scratch

To integrate client applications that use different protocols and message formats, create integration solutions by using applications, libraries, and integration projects.

Before you begin

If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see “Developing message flows” on page 482 and “Constructing message models” on page 2133.

About this task

In IBM App Connect Enterprise, you have different ways in which you can create an application:

- Create an application from scratch. For more information, see “Creating an application” on page 1963.
- Create an application by using patterns. For more information, see “Developing integration solutions by using patterns” on page 2056.
- Create an application that is based on a REST API Swagger document. For more information, see “Developing integration solutions by using REST APIs” on page 2011.
- Create an application that is based on WSDL and XSD files. For more information, see “Creating a solution based on WSDL or XSD files” on page 1967.
- Create an application that is based on an existing message set. For more information, see “Creating a solution based on an existing message set” on page 1968.

If you are not ready to create an application or library yet, you can use an integration project to hold resources in the meantime. When you are ready, you can either create an application or library and move the resources from the integration project, or you can convert your integration project to an application or library. To convert an integration project to an application or library, see “Converting existing projects to applications and libraries” on page 1970.

Procedure

For example, to create an application from scratch:

1. Create an application. An application is the container for resources that are required to develop your solution.
2. Optional: Create a library to contain a logical grouping of related code, schemas, message models, and other resources that can be reused. For more information, see “Creating a library” on page 1964.
3. Optional: Define the required library references in the application, as described in “Referencing resources in other libraries” on page 1976. To include one or more libraries of resources in your application, you create references to those libraries.
4. Create one or more message flows in your application to process your business data. For more information, see “Defining message flow content” on page 614, “Designing a message flow” on page 583, and “Creating a message flow” on page 574.
5. Use the default broker schema that is created when you create the message flow. Alternatively, create another broker schema. For more information, see “Creating a broker schema” on page 1966.
6. Create other resources that you need in the application and library. For more information, see “Creating resources in an application” on page 1964 and “Creating resources in a library” on page 1965.

What to do next

After you create your application, deploy and test the application. For more information, see How do I deploy and test ?.

Resource management overview

IBM App Connect Enterprise provides several types of container project that you can use to organize your message flow development resources.

When you develop message flows in the IBM App Connect Enterprise Toolkit, you can choose the type of container project, depending on your development style.

- An *application* is a container for the resources that are required to create a solution, and that should be deployed and managed together. Applications provide encapsulation, and resources within an application are packaged and deployed as a single unit, and can be managed as a single unit. For more information, see “Applications” on page 1943.
- A *service* is a special type of application that has a defined interface and implementation. Services provide assisted development to provide web services in an integration node. Like applications, services provide encapsulation and are deployed and managed as a single unit. For more information, see “Integration services” on page 1944.
- In IBM App Connect Enterprise, a *REST API* is a specialized application that can be used to expose integrations as a RESTful web service that can be called by HTTP clients. For more information, see “REST APIs” on page 2013.
- A *library* is a logical grouping of related code, data, or both. A library typically contains reusable helper routines and resources such as subflows, ESQL modules, message definitions, maps, and Java utilities. A library is useful to group resources of the same type or function, for reuse or ease of management. A library is also packaged and deployed as a single unit. You can use shared libraries and static libraries. A shared library can be used by applications. A static library can be used by applications or by independent message flows.

  If you use a static library to contain resources, each application that references that static library is deployed with its own private copy of that library. If a static library is updated, each application that references it must be redeployed with the updated static library. A shared library is deployed directly to an integration server. Any application can reference the resources in that deployed shared library. If that shared library is updated, the changes are immediately visible to all referencing applications. For more information, see “Libraries” on page 1945.
- An integration project is an unstructured container in which you can develop message flows and related resources. When resources from an integration project are packaged for deployment, you can choose exactly which resources should be packaged together. Integration projects cannot contain message models. For more information, see “Integration projects” on page 1959.

Containers in the Application Development view

Applications, integration services, REST APIs, libraries, and integration projects are shown in the Application Development view. Resources that are contained in an application, integration service, REST API, or library are also shown, whether the container refers to them directly or indirectly.

Message flow projects were replaced by integration projects in WebSphere Message Broker Version 8.0 (previously called Message Broker projects).

Applications, integration services, REST APIs, libraries, integration projects, and other types of project are represented by the following icons.

Icon	Description
	Application
	Integration service
	REST API
	Shared library

Icon	Description
	Static library
	Integration project
	Java project
	Message set project

Applications and libraries that have been deployed to an integration server are displayed directly beneath the integration server in the IBM App Connect Enterprise Toolkit and the web user interface. You can also view deployed applications and libraries by using the IBM Integration API or the **mqsilist** command.

Libraries that are referenced by applications are displayed beneath the application. Shared libraries are contained in the Referenced Libraries folder, and static libraries are contained in the Included Libraries folder.

You can also specify how an application or message flow is started after it is deployed, or after the integration node, integration server, or containing application is restarted. You can choose to start an application or flow manually, or for it to be started automatically. You can also choose to maintain the existing state of an application or library.

If you focus on a library or application, the Application Development view shows only that library or application. When you are viewing projects, you can also filter resources by using working sets.

For more information about applications, libraries, and integration projects, see the following topics:

- "Benefits of using applications and libraries" on page 1959
- "Runtime isolation and resource sharing with applications and libraries" on page 1960
- "Naming conventions in applications and libraries" on page 1962

For detailed instructions about how to manage your resources by using applications and libraries, see "Managing message flow resources" on page 572.

Applications

An *application* is a container for all the resources that are required to create a solution. An application can contain IBM App Connect Enterprise resources, such as flows, message definitions, libraries, and JAR files.

At design time, you define references to the following items:

- Zero or more message flow dependencies (such as a Java project or message set).
- Zero or more libraries that contain reusable resources.

You use applications to group all the resources that are required to deliver an integration solution, enabling easier development and management. If you are developing resources for multiple integration solutions in the IBM App Connect Enterprise Toolkit, then consider grouping your resources into applications. The use of a shared or static library helps organization by grouping reusable resources together. A shared library can also be used by other applications, services, shared libraries, or integration projects.

Applications typically contain message flows. If the message flow requires only one or two additional resources, such as one schema file and one ESQL file, consider storing all resources at the application level. But when you require multiple resources for your solution, or resources that might be shared with other solutions, consider putting the resources that your message flow requires into a shared library. That shared library can then be referenced by the application.

Applications provide runtime isolation whereby resources inside the application are not visible to other resources, such as message flows, libraries, or other applications that are running outside the application. Consider using applications if you need to ensure that updates to one group of deployed resources do not

affect another group. For example, use an application when you want to control which flows pick up the latest version of an ESQL module. For examples of how to use applications to achieve runtime isolation, see “Runtime isolation and resource sharing with applications and libraries” on page 1960.

Viewing applications

Applications are shown in the Application Development view. Resources that are contained in an application are also shown, whether an application refers to them directly or indirectly.

Applications are represented by the application icon  .

Applications that have been deployed to an integration server are displayed directly beneath the integration server in the IBM App Connect Enterprise Toolkit and the web user interface. Libraries that are referenced by applications are displayed directly beneath the application. Shared libraries are contained in a Referenced Libraries folder, and static libraries are contained in an Included Libraries folder. You can also view deployed applications by using theIBM Integration API or the **mqsilist** command; for example:

```
mqsilist integrationNodeName -e integrationServerName -k applicationName
```

This command specifies whether the application is running, and on which integration server. You can also see the deployed objects that are contained in that application, and a list of shared libraries that are referenced by the application.

You can also specify how an application or message flow is started after it is deployed, or after the integration node, integration server, or containing application is restarted. You can choose to start an application or flow manually, or for it to be started automatically. You can also choose to maintain the existing state of an application or library. You can also use the **mqsiapplybaroverride** command, as described in “Setting the start mode of message flows and applications at run time” on page 316.

If you focus on an application, the Application Development view shows only that application. When viewing projects, you can also filter resources by using working sets.

For more information about applications, see the following topics:

- “Benefits of using applications and libraries” on page 1959
- “Runtime isolation and resource sharing with applications and libraries” on page 1960
- “Naming conventions in applications and libraries” on page 1962

For detailed instructions about how to manage your resources by using applications, see “Managing message flow resources” on page 572.

Integration services

An integration service is a specialized application with a defined interface and structure that acts as a container for a web services solution.

You can integrate applications by using a service-oriented architecture (SOA). In an SOA, a service is often defined as a logical representation of a repeatable activity that has a specified outcome. Typically, a service is self-contained and its implementation is hidden from its consumers. To facilitate their reuse, services define a prescribed interface, which specifies how data is exchanged with the service.

In IBM App Connect Enterprise, an integration service is a specialized application with a defined interface that acts as a container for a web services solution:

- It contains message flows to implement the specified web service operations.
- The interface is defined through a WSDL file.

Like an application, an integration service can reference resources in shared libraries. The shared library is deployed before or with the integration service. If you update the resources in the shared library and redploy it, those changes are immediately available to the deployed integration service.

When you implement an integration service, you can deploy it to an integration server. You can start and stop the deployed service as you would an application. A web service consumer can interrogate the deployed service to return its interface.

Interfaces

An *interface* consists of one or more operations and a binding style.

An *operation* is a description of an action that is implemented by the service. Each operation can have either of the following types:

- *Request-response* type operations mean that a request is sent and a response returned to the interface.
- *One-way* type operations mean that only a request is sent, and no response is needed.

Each operation in the interface defines the data that can be passed in the form of inputs to and outputs from the component when the operation is called. A one-way operation has only an input. A request-response operation has both an input and output. Each operation can have one or more faults defined to handle error conditions.

The *binding style* specifies the protocol and data format of the operation.

Integration service API

You can generate a JavaScript client API from an existing integration service. The JavaScript client API provides operation functions that you can call from a program that is running in a JavaScript environment.

Libraries

A *library* is a logical grouping of related code, data, or both. A library typically contains reusable helper routines and resources such as subflows, ESQL modules, message definitions, maps, and Java utilities. You can use a library to group resources of the same type or function, and to aid the management and reuse of such resources

Consider using libraries for the following functions:

- To group common types of resource (such as all your ESQL routines)
- To group resources by function (such as all your error-handling routines)
- To share routines and definitions across multiple teams or projects
- To use different versions of a coherent set of routines and definitions

Two types of library exist in IBM App Connect Enterprise: *shared libraries* and *static libraries*.

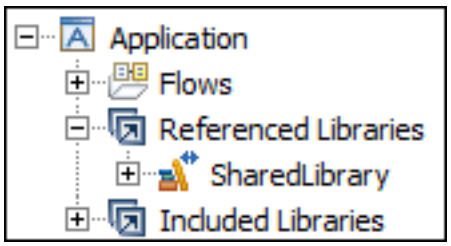
Consider a scenario when you want to develop a set of common resources and make them available to multiple applications.

Shared libraries
: If you want to deploy and manage just one copy of those common resources, use a shared library. A shared library can be deployed directly to an integration server. Any application can reference the resources in that deployed shared library. If that shared library is updated, the changes are immediately visible to all referencing applications.

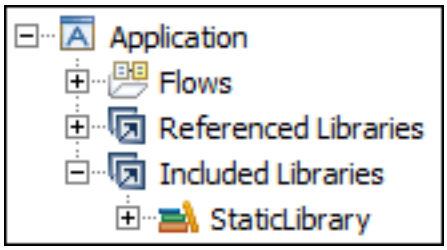
Static libraries
: If you want each application to use a different version of the contained resource, use a static library. Each application that references that static library is deployed with its own private copy of that library. If a static library is updated, each application that references it must be repackaged and redeployed with the updated static library.

Static libraries are represented by the static library icon  . Shared libraries are represented by the shared library icon  .

A shared library that is referenced by an application or another shared library is shown in the Referenced Libraries folder of the application or shared library.



A static library that is referenced by an application or another static library is shown in the Included Libraries folder of the application or static library.



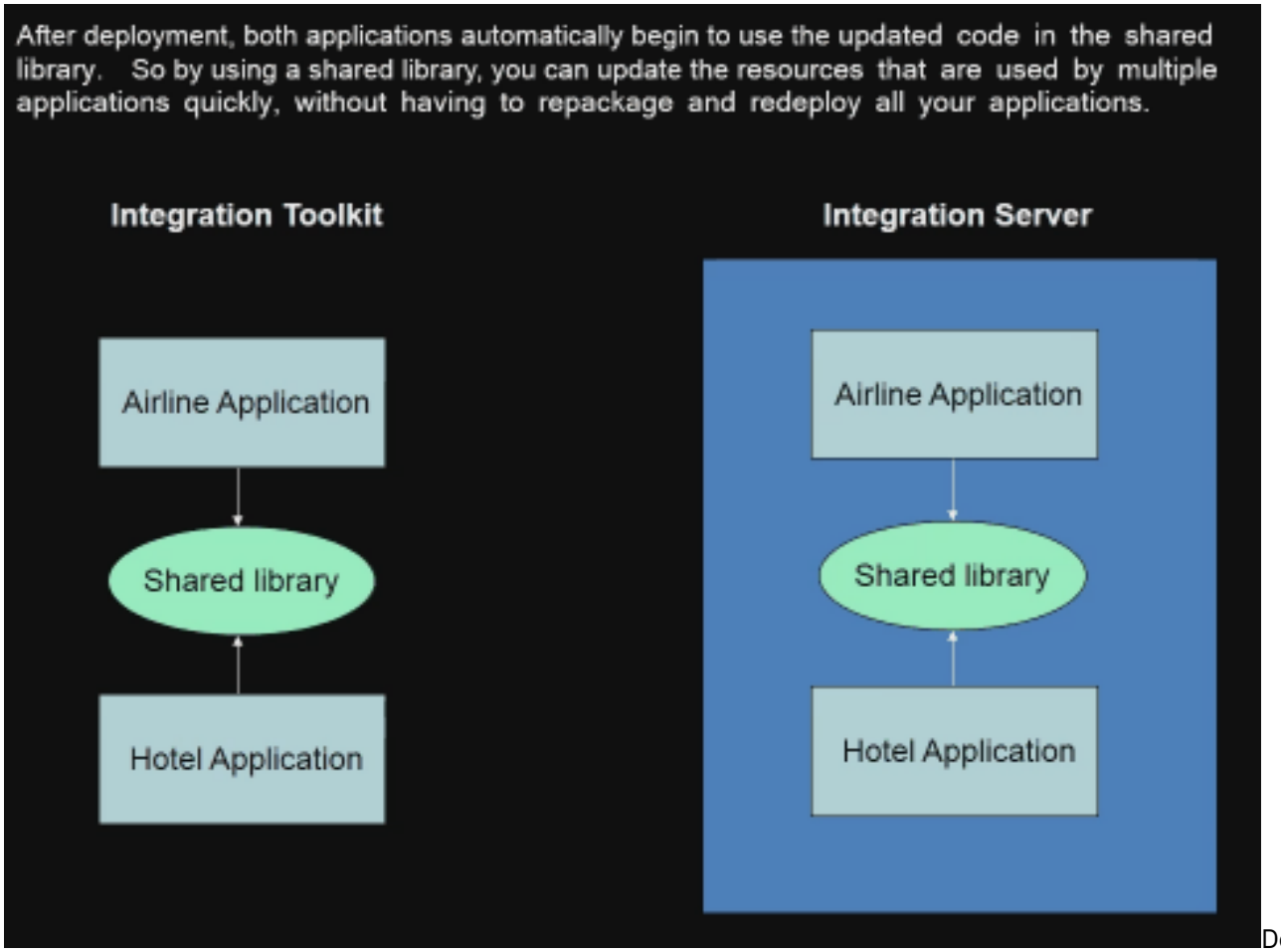
A shared library can refer to other shared libraries only. Similarly, a static library can refer to other static libraries only.

Libraries that are deployed to an integration server are displayed directly beneath the integration server in the IBM App Connect Enterprise Toolkit and the web user interface. Libraries that are referenced by applications are displayed directly beneath the application in the IBM App Connect Enterprise Toolkit. You can also view deployed libraries by using theIBM Integration API or the **mqsilist** command, for example:

```
mqsilist integrationNodeName -e integrationServerName -y sharedLibraryName
```

In this example, the command lists the resources that are deployed in a shared library, the integration server to which that shared library is deployed, and any applications that refer to the shared library.

The following animation illustrates the difference between a static and shared library.



Download mp4 file

The following table summarizes the differences between static and shared libraries.

	Shared library	**Static library**
Support for libraries	Shared libraries are introduced in IBM Integration Bus 10.0.	The libraries that were introduced in WebSphere Message Broker Version 8.0 are renamed as static libraries.
Referencing libraries	If multiple applications reference a shared library, when the applications are deployed, each application uses the artifacts directly from the deployed shared library.	If multiple applications reference a static library, when the applications are deployed, each application has its own private copy of the library and the resources that are contained in it.
Updating libraries	If you update and redeploy a shared library, all applications that reference that shared library see the updates automatically.	If you update a static library, you must repackage and redeploy each application that references that library, unless applications need to use different versions of that library.

	Shared library	Static library
Deploying libraries	Shared libraries can be deployed directly to the integration server, or they can be deployed in the same BAR file as the applications that reference them. If a shared library is deployed in a BAR file, it can still be used by applications or shared libraries in other deployed BAR files.	Static libraries are packaged and deployed in the same BAR file as the applications that reference them.

For more information, see the following topics:

- "Shared libraries" on page 1948
- "Static libraries" on page 1958

Shared libraries

You might want to develop a set of common resources and make them available to multiple applications. If you want to deploy and manage just one copy of those common resources, use a shared library.

A shared library can be deployed directly to an integration server. Any application can reference the resources in that deployed shared library. If that shared library is updated, the changes are immediately picked up by all referencing applications. (If you use a static library to contain resources, each application that references that static library is deployed with its own private copy of that library. If a static library is updated, each application that references it must be repackaged and redeployed with the updated static library.)

A shared library must be deployed with or before an application that references it. If you deploy resources by dragging them onto an integration server, deploy the shared library first. If you deploy resources by adding them to a BAR file, ensure that you include any shared libraries that your application references. Similarly, you cannot remove a deployed shared library while deployed applications are referencing it.

The following conditions apply to the referencing of shared libraries:

- Shared libraries can reference other shared libraries but cannot reference static libraries.
- Static libraries and integration projects cannot reference shared libraries.
- If an application references more than one shared library, the application has visibility of the resources in all of the shared libraries, but each shared library does not automatically have visibility of the resources in the other shared libraries.
- If a shared library (A) references another shared library (B), shared library A has visibility of the resources in shared library B, but shared library B does not have visibility of the resources in shared library A.
- If a shared library (A) references another shared library (B), shared library B cannot create a reference to shared library A.

The shared library hierarchy is preserved across development, deployment, and operational management processes. For example, you can see the shared library hierarchy in the IBM App Connect Enterprise Toolkit and the BAR File editor. You can also view deployed libraries by using the IBM Integration API, or the **mqsilist** command with the -y option; for example, using the command with an integration node:

```
mqsilist integrationNodeName -e integrationServerName -y sharedLibraryName
```

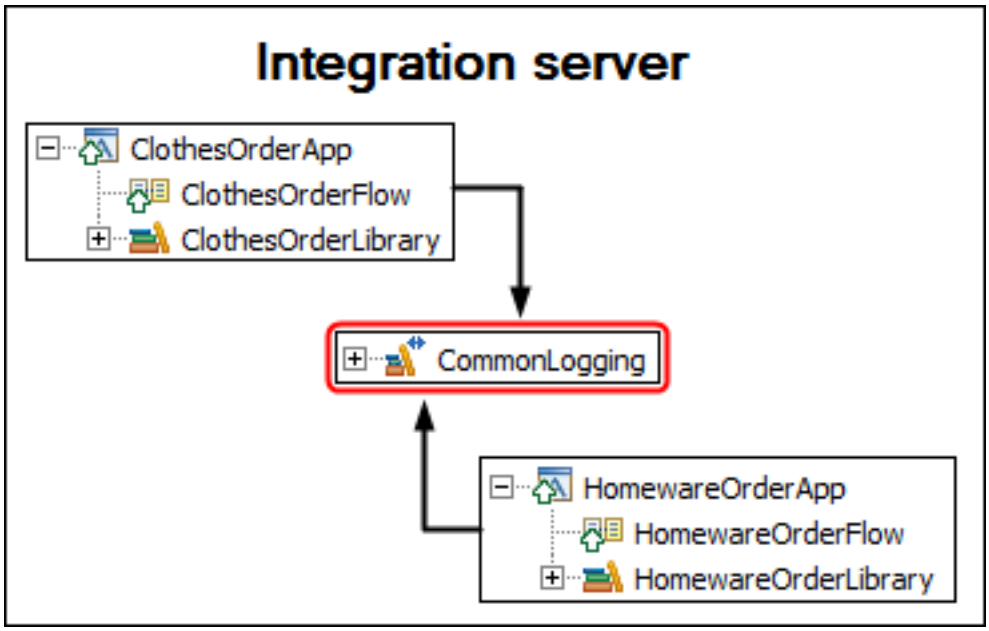
In this example, the command lists the resources that are deployed in a shared library, the integration server

The following examples demonstrate when you might use shared libraries.

Example 1: Sharing common logging code

You build a shared library that contains subflows and ESQL code that provides common logging functions. Multiple applications explicitly reference the shared library so that they can use the common logging functions. These applications are built and deployed.

You then update the shared library to ensure that additional information is recorded to the log. When you redeploy the shared library, all the deployed applications that explicitly reference this shared library pick up the changes automatically and start to log the additional information.



Example 2: Sharing a set of XML or DFDL schema files

You build a shared library that contains a logical set of complex DFDL or XML schema files, such as those for the Health Level Seven (HL7) standard. Multiple applications explicitly reference this shared library so that they can use the HL7 schema files for Mapping nodes or XMLNSC validation. These applications are built and deployed.

When the shared library is deployed, memory usage can be affected because the complex schema files must be compiled into a model. However, you reduce this memory usage when you update and redeploy the applications that reference the shared library because the schema files do not need to be recompiled.

Example 3: Mapping between clashing XML or DFDL schema files

You might want to map between different versions of an XML or DFDL schema. Both versions might declare the same elements and types, which cannot coexist in one application.

You can build one shared library that contains version 1 of a schema file, and another shared library that contains version 2. You can then build a map that references the schema files in the shared libraries and maps between the two versions.

At deployment time, separate models are built for each of the shared libraries. Therefore, no clash results from the duplicated elements and types.

Resources that are not supported in a shared library

The following resources are not supported in shared libraries:

- Message flows
- CORBARequest node and IDL files
- SCA nodes

- WebSphere Adapters nodes
- XSLTransform node
- MRM message sets
- XML files
- XSL files

The following sections describe how to use supported resources in shared libraries:

- "XML or DFDL schema files" on page 1950
- "WSDL files" on page 1950
- "ESQL files" on page 1950
- "Java files" on page 1951
- "Subflows" on page 1951

XML or DFDL schema files

When you validate against deployed XML or DFDL schema files in shared libraries, you must specify the name of the shared library that contains the appropriate compiled model. For example, when you create a message flow, you can configure the input node to validate against the message model for a shared library that is called HL7.

For more information, see "Shared libraries and XML or DFDL schema files" on page 1953.

WSDL files

When you validate against deployed WSDL files in shared libraries, you must specify the name of the shared library that contains the appropriate compiled model. You can use a WSDL file in a shared library to configure a SOAPInput, SOAPRequest, or SOAPAsyncRequest node. When you drag a WSDL file from a shared library onto one of these nodes, the `WSDL file name` property specifies the name of shared library; for example:

```
CurrentAccount.wsdl in Shared Library Shlib1
```

The `Message model` field is also set to the name of the top-level shared library that contains the WSDL file inside braces {}; for example: `{Shlib1}`. The WSDL might be contained in this specific shared library, but might be contained in one of the shared libraries that is referenced by this library. To drag a WSDL file from a shared library onto a SOAP node, the container of the message flow must reference the shared library that contains the WSDL file.

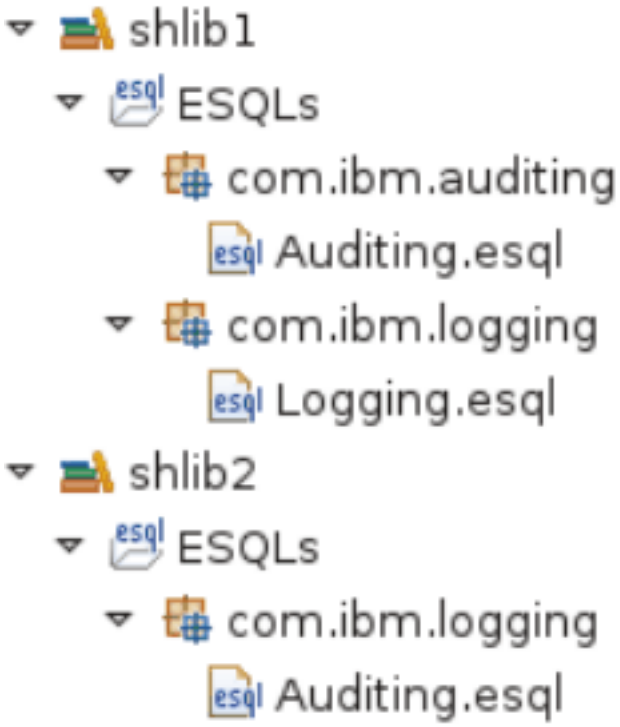
Multiple applications can access the same WSDL file in a single shared library. The shared library that contains the WSDL file must be deployed with or before the applications that refer to it. If you update the WSDL file and redeploy the shared library, those changes are immediately available to all referring applications.

A WSDL file of the same name can exist in multiple containers. For example, WSDL files with the same name can exist in an application and a shared library that is referenced by that application. You can also use multiple SOAPInput nodes in a message flow that reference different WSDL files with the same name in referenced shared libraries. A WSDL file in one shared library can use a schema file in another referenced shared library.

ESQL files

To avoid clashes between ESQL files in an application and its referenced shared libraries, the broker schema for the ESQL files is used. When you store ESQL files in a shared library, you must place the ESQL files inside a schema that is not the default empty schema.

Each shared library can have one or more ESQL files in one or more ESQL schemas. However, no two libraries in the same scope can have ESQL files in the same schema.



ESQL code in applications or shared libraries can also call static Java methods in referenced shared libraries by using the shared library qualifier in the CLASSLOADER clause. For more information, see Java routine example 4 in CREATE PROCEDURE statement.

Java files

You can store Java files in shared libraries. JAR files are packaged inside shared libraries. Java classes that are deployed in a shared library are not available to the integration server wide class loader. When you deploy a shared library that contains Java files, a new classloader is created for that shared library.

Java classes in the integration server wide classloader are isolated from the Java classes in any shared library classloaders. Similarly, Java classes in shared library classloaders are isolated from Java classes in the integration server wide classloader.

Java classes in applications, static libraries, or independent projects cannot access Java classes in shared libraries.

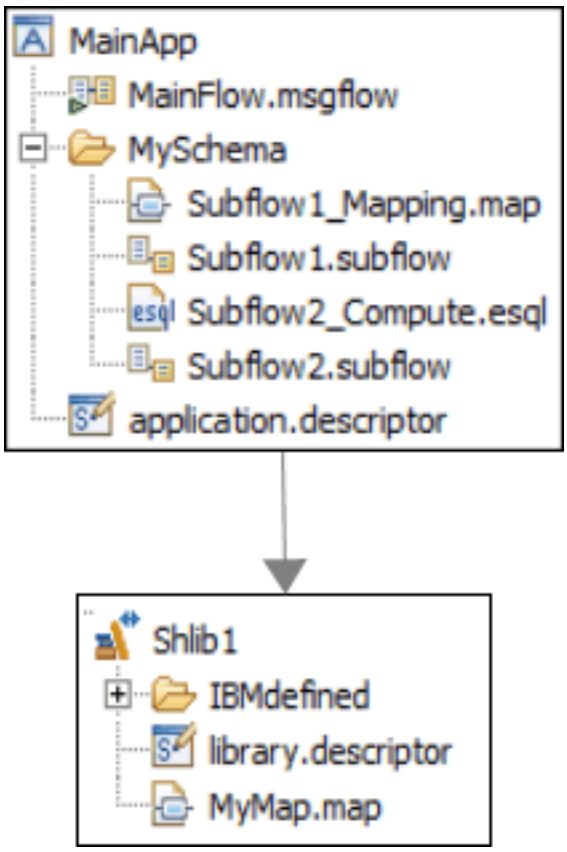
For more information about storing Java files in shared libraries, see “Shared libraries and Java files” on page 1954.

Subflows

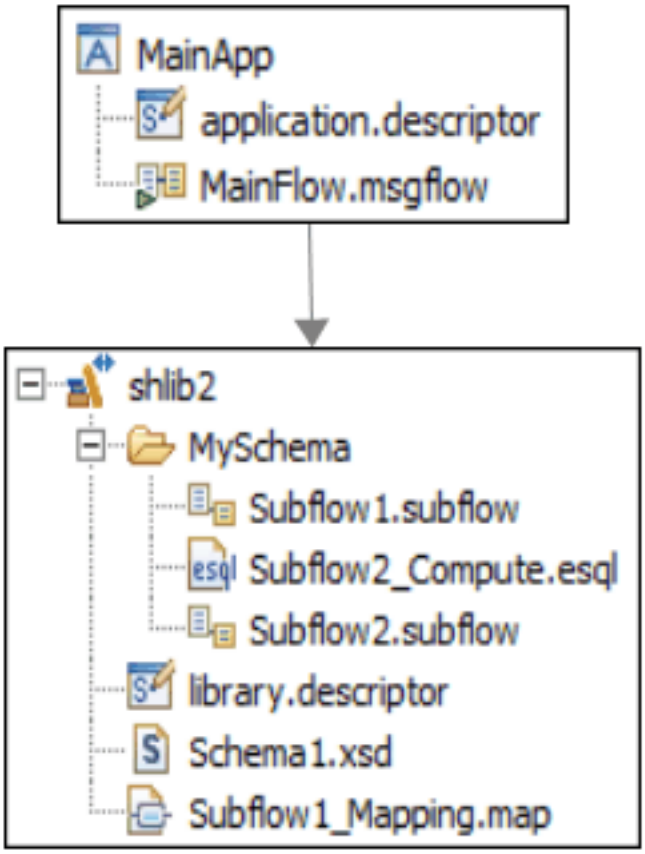
When you store subflows in a shared library, you must place the subflows inside a schema that is not the default empty schema.

When an application or shared library references other shared libraries, all the subflows for a broker schema must be in a single container. Subflows for a broker schema must not be in both an application (or shared library) and a shared library that is referenced by that application (or shared library). Subflows for a broker schema must not be in two or more shared libraries that are referenced by a single application or shared library. All the subflows in a broker schema must be either in the main application or shared library, or in a single referenced shared library.

Example 1: All subflows in the same broker schema are stored in the main application.



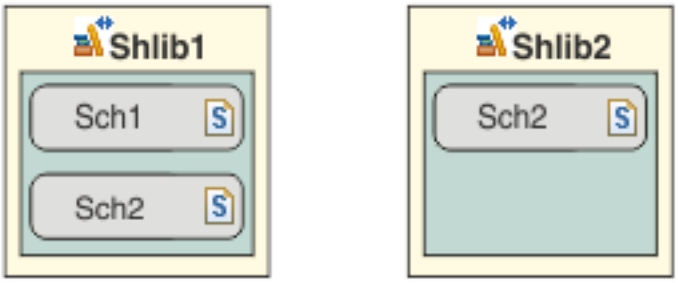
Example 2: All subflows in the same broker schema are stored in a referenced shared library.



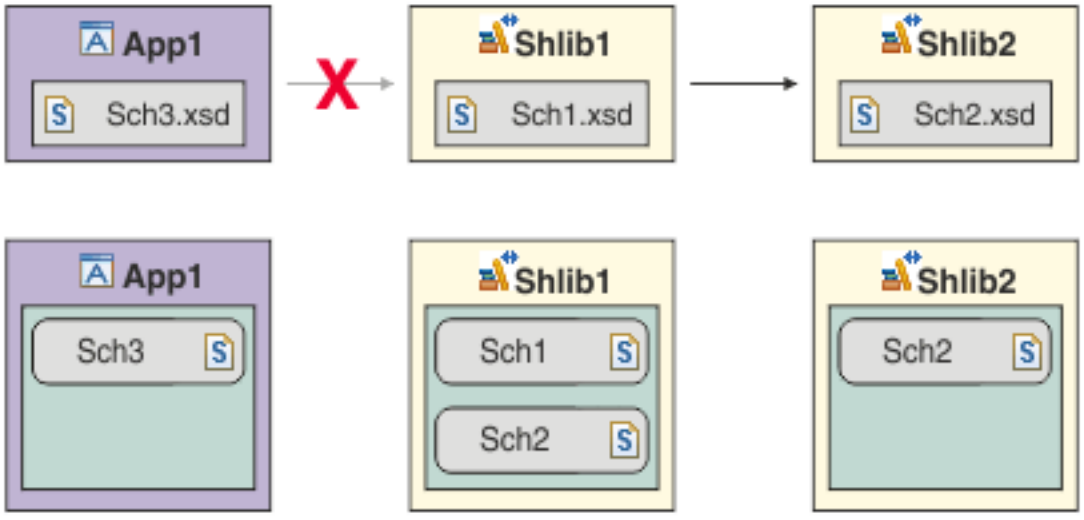
Shared libraries and XML or DFDL schema files
When you validate against deployed XML or DFDL schema files in shared libraries, you must specify the name of the shared library that contains the appropriate compiled model.

Shared libraries can contain XML or DFDL schema files. Shared libraries can also access schema files that are contained in referenced shared libraries, either by using the import or include options.

When the first shared library is deployed, a model is compiled that contains all the schema files in that shared library and any referenced shared libraries.



The application or applications that reference the shared libraries can also contain their own schema files. A separate model is compiled that contains only the schema files from the application. The schema files in the application cannot access schema files in a shared library.



To validate against the deployed schema files, the compiled representation of the XML schemas is passed to the parser. Therefore, you must specify the name of the shared library that contains the appropriate compiled model. For example, when you create a message flow, you can configure the input node to validate against the message model for a shared library that is called hl7 by specifying a value of {hl7} in the **Message model** property on the **Input Message Parsing** panel of the node. To access the **Message model** value in the Validate node, you can set the **MessageSet** field of the Properties tree by including the following ESQL statement in a Compute node:

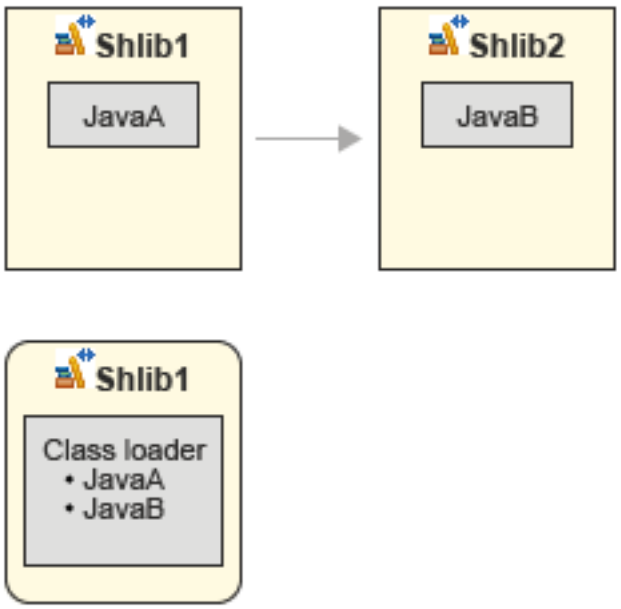
```
SET OutputRoot.Properties.MessageSet = '{Shared LibraryName};'
```

You can use the following methods to view the relationships between applications and shared libraries:

- The **mqsilist** command
- User trace and system trace
- The web user interface
- The Integration Explorer view of the IBM App Connect Enterprise Toolkit

Shared libraries and Java files
You can store Java files in shared libraries. JAR files are packaged inside shared libraries.

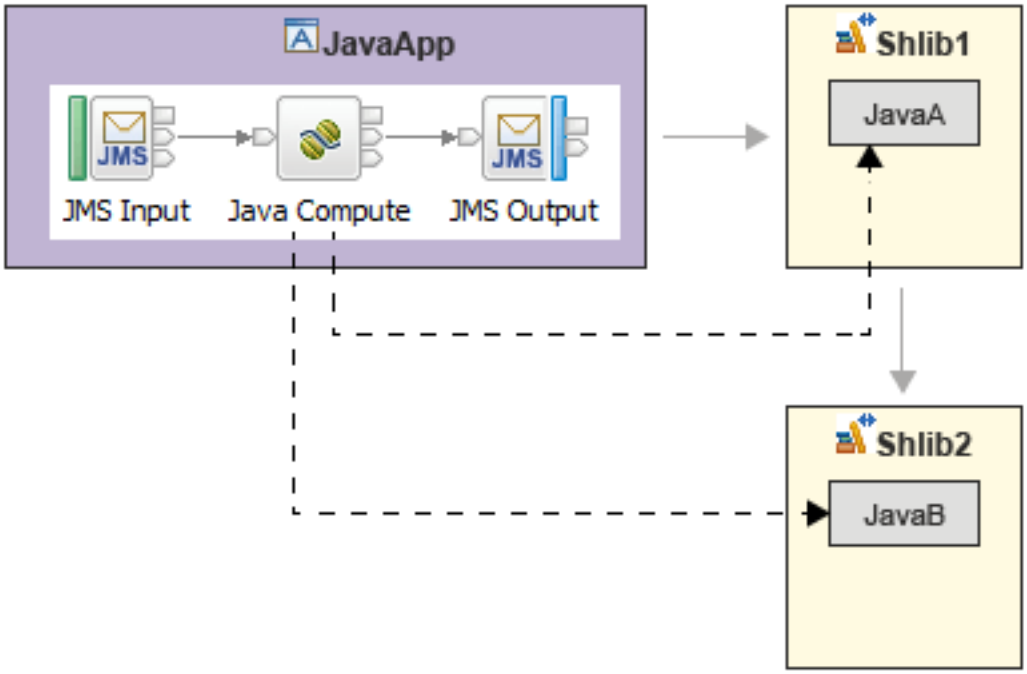
Java classes that are deployed in a shared library are not available to the integration server-wide class loader. When you deploy a shared library that contains Java files, a new class loader is created for that shared library. This class loader contains all Java classes in the shared library, as well as the Java classes from all referenced shared libraries. Delegation does not exist from one shared library class loader to another shared library class loader. The following diagram shows that a shared library, `shlib1`, contains a Java file called `JavaA`. Shared library `shlib1` refers to another shared library, `shlib2`, which contains a Java file called `JavaB`. When `shlib1` is deployed, a class loader is created for the Java files in that shared library as well as the referenced shared library.



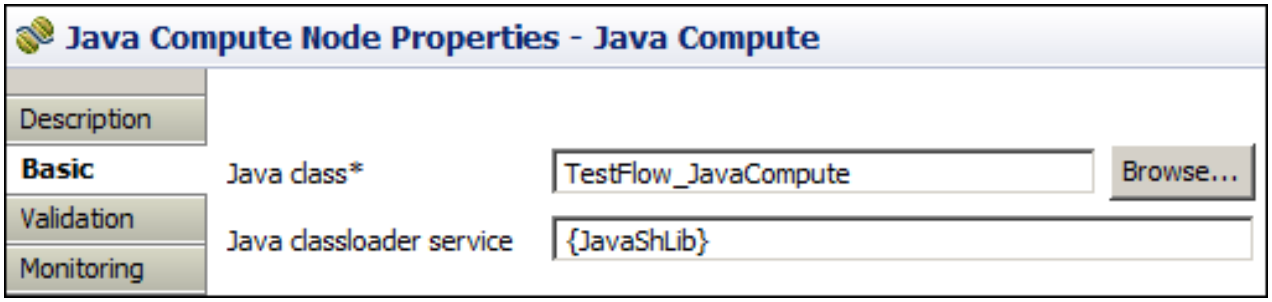
A Java class can exist in multiple class loaders if that Java class is contained in a shared library that is referenced by other shared libraries. When you update that Java class by redeploying the shared library that contains it, all class loaders for all shared libraries that contain that Java class are deleted and re-created.

Java classes in the integration server-wide class loader are isolated from the Java classes in any shared library class loaders. Similarly, Java classes in shared library class loaders are isolated from Java classes in the integration server-wide class loader.

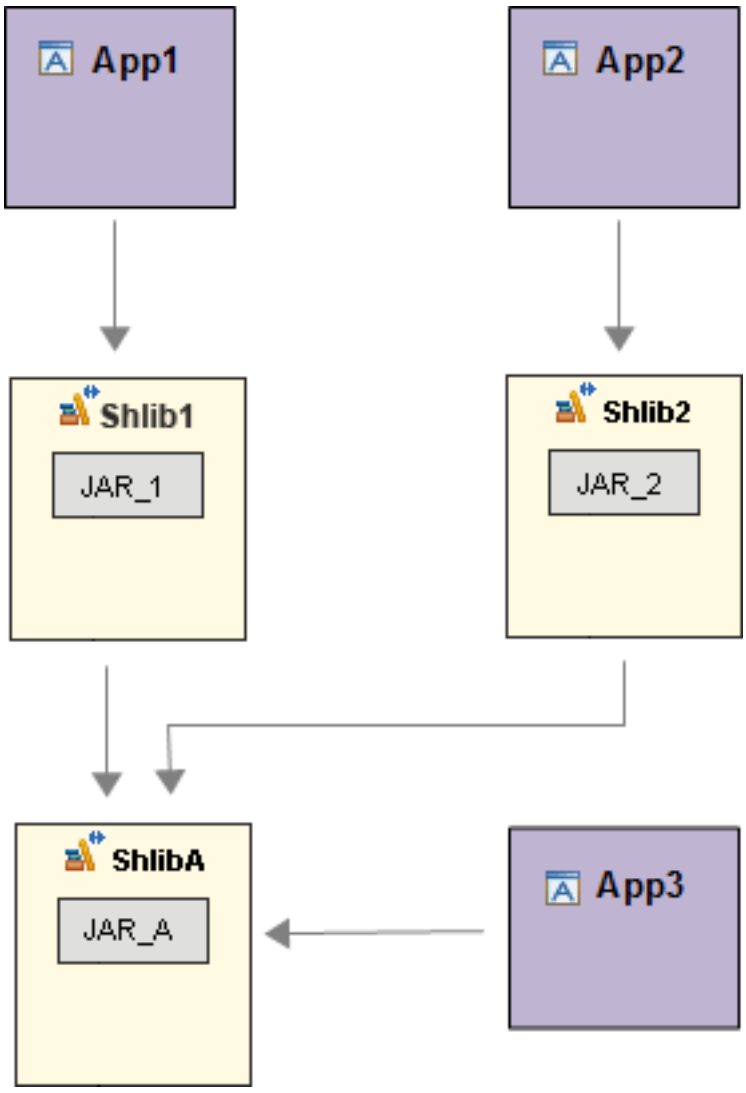
A JavaCompute node in a message flow or subflow can access the Java classes in a shared library and any referenced shared libraries. The following diagram shows a JavaCompute node in a message flow in an application, but the JavaCompute node could be in a subflow in a shared library.



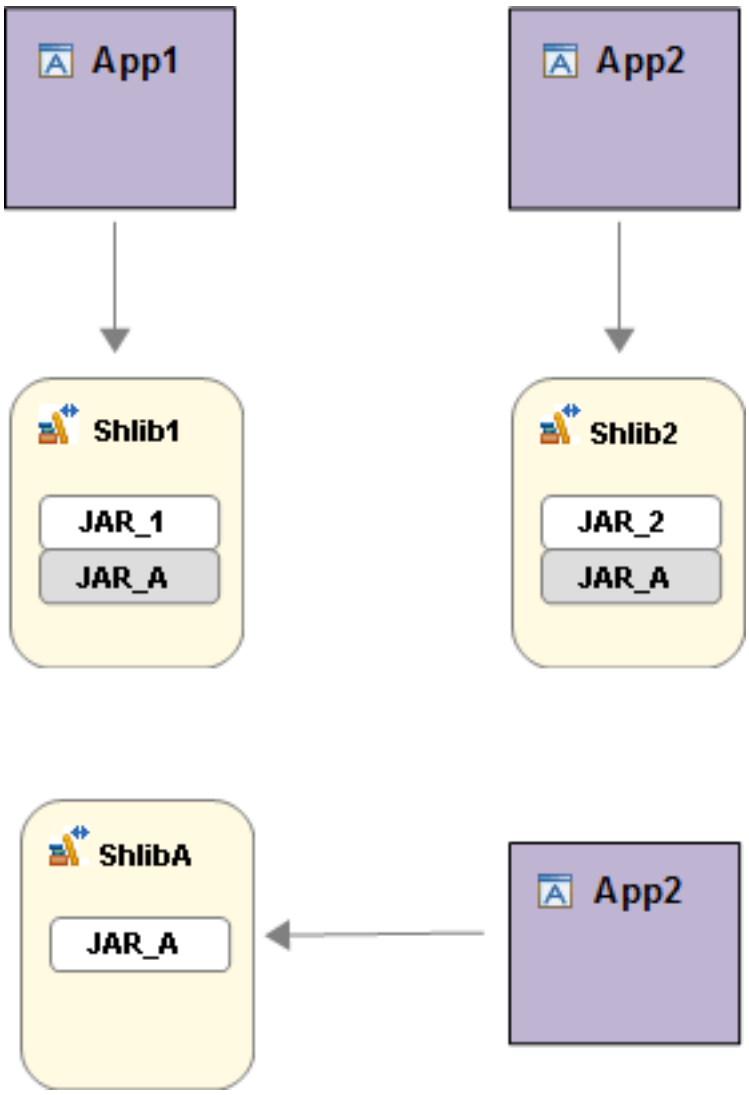
To access these Java classes, you specify a shared library qualifier for the Java Class Loader policy on the JavaCompute node. The shared library qualifier is inside braces {}, as shown in the following example:



When you refer to a shared library that contains Java classes, all of the Java classes might be loaded into multiple class loaders. The Java classes can be loaded into the class loader for that shared library, and the class loader for any other shared libraries that refer to that shared library. The following diagram shows how three different applications refer to three shared libraries. Application 1 refers to shared library 1, which also refers to shared library A. Application 2 refers to shared library 2, which also refers to shared library A. Application 3 refers to shared library A only.



The following diagram shows how the applications access the Java classes at run time. Shared libraries 1 and 2 have their own loaded copy of the classes from JAR_A in memory. Applications 1, 2, and 3 all see different copies of JAR_A.



For applications 1 and 3 to access the same loaded copy of the classes from JAR_A, application 1 would need to use the shared library qualifier for shared library A. Static variables in a shared library are duplicated for each shared library qualifier. JavaCompute nodes access the same static variables by using the same shared library qualifier. However, when you redeploy a shared library, the Java class loaders for that shared library, and any other shared libraries that reference that shared library, reload the Java classes.

ESQL code in applications or shared libraries can also call static Java methods in referenced shared libraries by using the shared library qualifier in the CLASSLOADER clause. For more information, see Java routine example 4 in CREATE PROCEDURE statement.

Here are some more examples of possible scenarios:

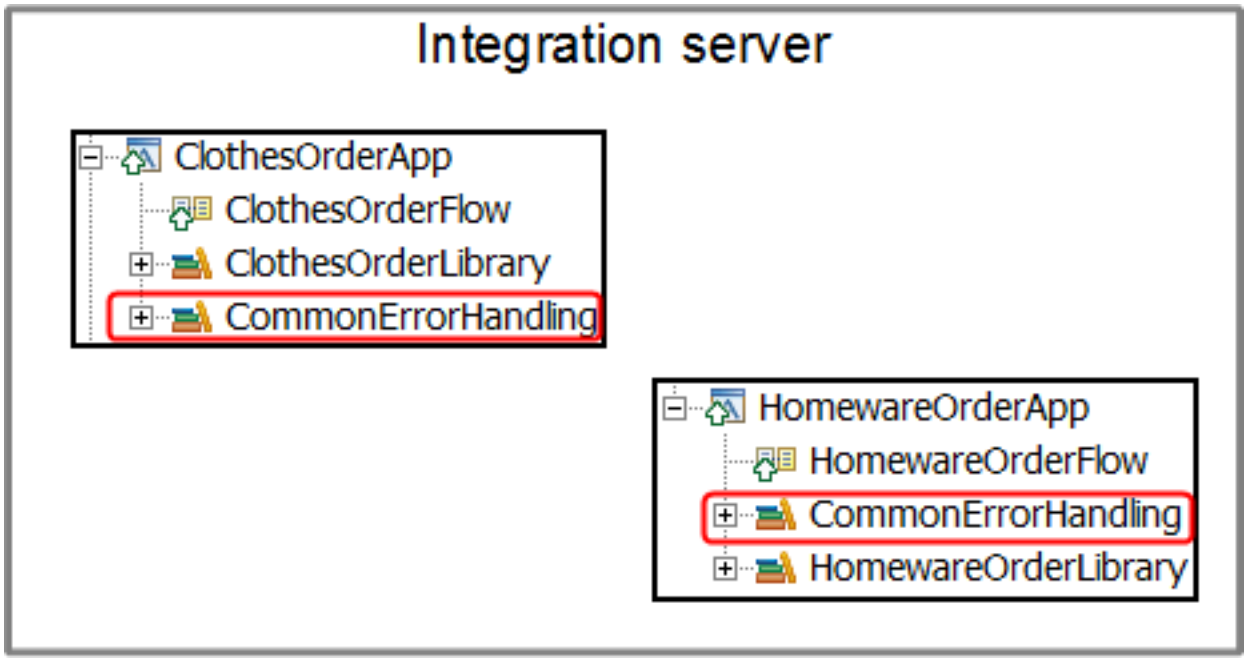
- A JavaCompute node in a subflow in a shared library can access the Java classes in the same shared library or in referenced shared libraries.
- If a message flow contains multiple JavaCompute nodes, each JavaCompute node can refer to a different shared library.
- Multiple applications can use the Java classes in a single shared library.

Java classes in applications, static libraries, or independent projects cannot access Java classes in shared libraries.

Static libraries

The libraries that were introduced in WebSphere Message Broker Version 8.0 are static libraries.

A static library can be referenced by one or more applications. Changes that are made to the library in the IBM App Connect Enterprise Toolkit are available to all applications that reference that library. However, when the applications are packaged into a BAR file and deployed, each application has its own private copy of the library and the resources that are contained in it. If you update a static library, you must repackage and redeploy each application that references that library.



Static libraries can be referenced by applications, services, integration projects, or other static libraries. You cannot refer to a static library from a shared library. Similarly, you cannot reference a shared library from a static library.

Static libraries behave in the following ways, depending on the resources that reference them.

- If a static library is referenced by an application, a copy of that referenced library is isolated at run time from resources outside the application. This copy of the library is not available to resources outside the application.
- If a static library is referenced by a service, a copy of that referenced library is isolated at run time from resources outside the service. This copy of the library is not available to resources outside the service.
- If a static library is referenced by an independent resource, that referenced library is available to all independent resources that are deployed in the same integration server as the library.

Some restrictions are associated with static libraries. You can overcome the following restrictions by using shared libraries.

- When an application is deployed, all the XML and DFDL schema files in that application and its referenced static libraries are compiled into a single model that represents the application. Duplicate element or type names cannot coexist in a single model. Therefore, an application cannot include conflicting XML or DFDL schema files. However, you might need to use multiple conflicting XML or DFDL schema files in a single application. For example, you might want to map elements between versions one and two of a particular XML schema file.
- The presence of complex XML schema files in static libraries can affect the compilation process and the time that is taken to deploy the application. Static libraries are redeployed with the application. Therefore, every time the application is deployed, all the XML schema files are recompiled, even if they have not changed.

Integration projects

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

Message flow projects have been replaced by integration projects in IBM Integration Bus 10.0.

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

- Message flows
- Subflows
- Message maps
- ESQL files
- Database definitions
- BAR files
- Test Clients

You cannot create schema files in an integration project; you can create them in an application or library only.

An integration project can reference a static library but not a shared library.

Integration projects are shown in the Independent Resources folder in the Application Development view. Integration project resources are created as files, and are displayed under the project. These resources define the content of the message flow, and additional objects that contain detailed configuration information. For example, a project might contain ESQL modules or message mappings, used by one or more nodes in the message flow.

You can add resources to an integration project and deploy those resources directly from the integration project, without having to deploy them in an application or library.

You can share your Integration projects with others by exporting and importing an integration project as a compressed file known as a *project interchange file*. You can also copy project directories from your workspace directory to another location; for example, to backup the project directory or to use in another workspace directory.

Benefits of using applications and libraries

Use applications and libraries to help manage multiple solutions and to share resources across multiple teams or projects.

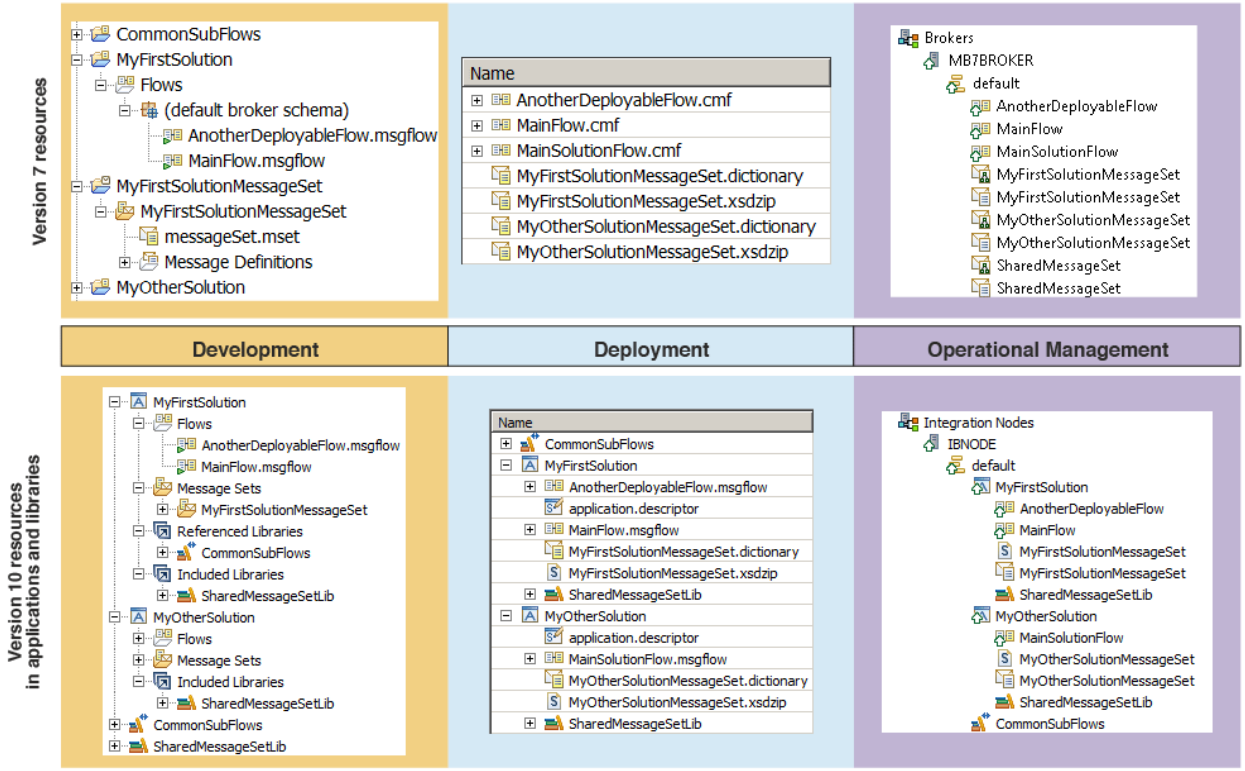
The use of applications and libraries has the following benefits:

- Flexible organization of resources.
  - You can organize resources that are required by message flows (such as XSD files, IDL files, WSDL, and `.inadapter` or `.outadapter` components) into libraries.
  - You control the structure of the library.
  - You can build a library by using resources from several projects (for example, a message set, and a Java project).
- Encapsulated packaging of flows and their dependencies.
  - Applications are packages of message flows and libraries.
  - You can use message flows from several projects to build an application.
  - You can deploy and redeploy a whole application.
- Modular and collaborative development.
  - Libraries can reference other libraries to reuse resources.

- Unified packaging and organization through application development, deployment, and operational environments.
  - You can develop, deploy, then administer applications.
  - You can administer individual libraries.
  - You can maintain dependency relationships during operation.

Consistency through the development, deployment, and operational management processes

Applications, libraries, and integration projects provide a consistent view of your resources through the development, deployment, and operational management processes. In WebSphere Message Broker Version 7.0 and earlier versions, file types and concepts were different at each stage of the development process. Multiple project types were used in the IBM App Connect Enterprise Toolkit to contain different types of resource. No consistent method existed to reuse common components across all processes. However, applications and libraries span the IBM App Connect Enterprise Toolkit for development, the BAR file for deployment, and tools like the web user interface for operational management.

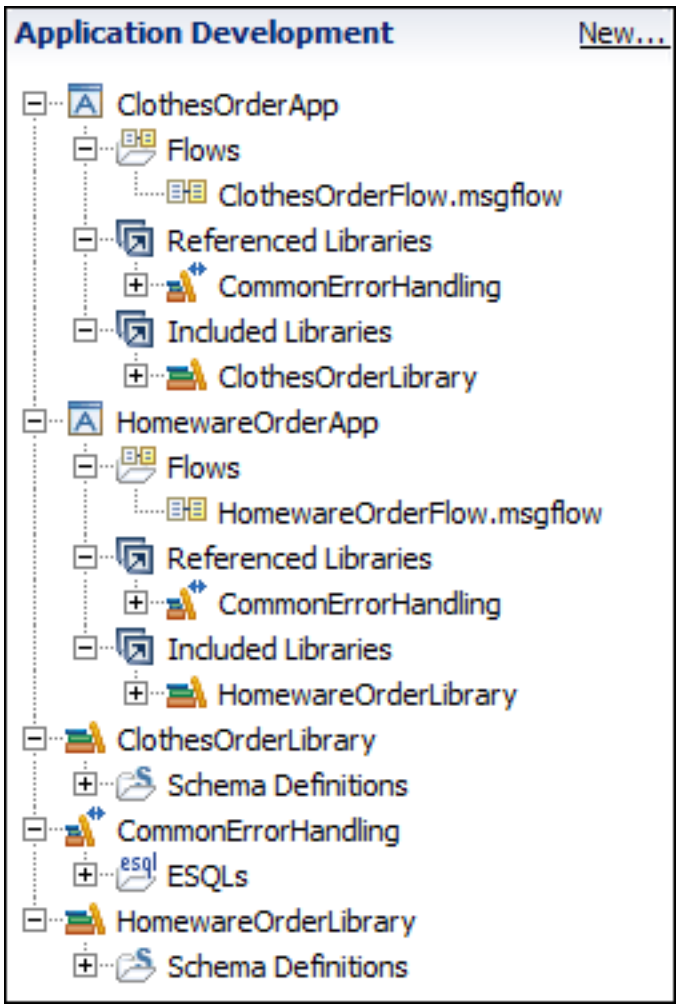


Runtime isolation and resource sharing with applications and libraries

You can use applications to encapsulate resources for a solution or to provide runtime isolation. You can use libraries to group common resources or share routines and definitions between teams, projects, or integration nodes.

Applications provide runtime isolation whereby resources inside the application are not visible to other resources, such as message flows, libraries, or other applications that are running outside the application. Consider using applications if you need to ensure that updates to one group of deployed resources do not affect another group. For example, use an application when you want to control which flows pick up the latest version of an ESQL module.

The following example illustrates how you can use applications to contain separate solutions, and libraries to contain shared error-handing code.



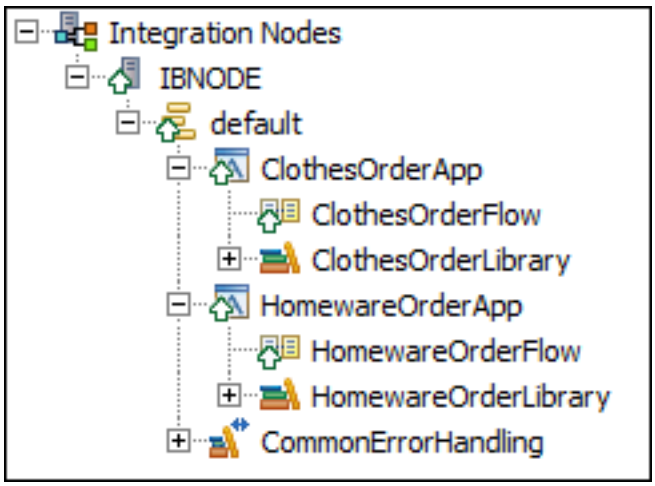
The ClothesOrderApp application contains the following resources:

- A message flow called ClothesOrderFlow
- A reference to a static library called ClothesOrderLibrary, which contains XSD schema files that are specific to the order type
- A reference to a shared library called CommonErrorHandling, which contains some common error-handling ESQL code

The HomewareOrderApp application contains the following resources:

- A message flow called HomewareOrderFlow
- A reference to a static library called HomewareOrderLibrary, which contains XSD schema files that are specific to the order type
- A reference to a shared library called CommonErrorHandling, which contains some common error-handling ESQL code

The following diagram shows the resources in the Integration Explorer view after they are deployed. The shared library is deployed with or before the applications that reference it.



The shared library appears at the same level as the applications that reference it. The static libraries are shown nested under the applications that reference them. Resources in the applications and their referenced static libraries are not visible to other resources outside those applications.

Applications provide this isolation at run time. This isolation also applies if a resource that is contained in an application is also deployed separately to the same integration server.

Assume that a static library is deployed to an integration server with a message flow that references that library. The same static library is also contained in an application that is deployed to the same integration server. If that static library is updated in the IBM App Connect Enterprise Toolkit and redeployed, the message flow that is deployed to the integration server can see the changes. However, the application cannot see the changes. For the application to pick up the updated static library, you would need to rebuild and redeploy the BAR file that contains the application.

Naming conventions in applications and libraries

When you add resources to applications and libraries, ensure that you use appropriate naming conventions to avoid conflicts. For example, you can qualify a file path with a library name, or differentiate a schema element by using a QName.

A *QName* refers to a namespace-qualified element from a message model, which can be an XML schema, DFDL schema file, or a message set. These QNames must be unique throughout an application or library and any of their referenced libraries. For example, "Application1" contains two schema files, `schemaOne.xsd` and `schemaTwo.xsd`. Both schema files contain the element "sameElement", which is qualified by the namespace `http://mynamespace.com`. This naming convention causes a conflict because both schema files have the same QName for the element named "sameElement", which is qualified by the same namespace.

Both schema files also contain the element "diffNSElement", but this element is qualified by a different namespace in each schema file. Therefore, no conflict occurs. If a conflict of QNames is detected, each XML or DFDL schema file, or message set, that contains a construct with that QName is marked with an error in the Application Development view and Problems view.

The broker schema defines the relative path from the application or library to the resource name. The following resource names must be unique in an application or library and all referenced libraries:

- An XSD resource name and path pair
- A WSDL resource name and path pair
- A message flow resource name and path pair
- An adapter resource name and path pair (`.inadapter` and `.outadapter`)
- A CORBA IDL resource name and path pair
- A message map resource name and path pair (`.map`)
- An XSLT resource name and path pair

If a conflict of a resource name and path is detected, each resource is marked in the Application Development view and Problems view. If a conflict of a file resource and path is detected, each resource is marked in the Application Development view and Problems view.

You cannot reference `.inadapter` and `.outadapter` files for the same SAP operations in separate libraries in the same application. Instead, you must use one of the following configurations:

- Two separate applications and adapter libraries for inbound and outbound operations
- A single adapter library with both inbound and outbound adapters that are generated in different namespaces and used in two separate applications. In this configuration, one application must have an inbound flow, and the other must have an outbound flow.
- A single adapter library with both inbound and outbound adapters. These adapters must be generated in different namespaces and used in one application with both inbound and outbound flows.
- Two separate applications that have their own business object schema, adapter, and flows.

Creating an application

Create an application to contain all the resources that are required to develop your solution.

Before you begin

Read concept information about applications in "Applications" on page 1943.

About this task

The order in which you create resources is flexible. You can begin by creating an application, then create resources to add to it, or you can begin by creating libraries of resources, then create an application that refers to them. You can also convert an existing project into an application.

The resources that an application can contain include message flows, ESQL files, and subflows. An application can also refer to one or more libraries. The resources that a library can contain include Adapter connections, CORBA IDL files, and schemas. To create a library of resources, see "Creating a library" on page 1964.

To create an application, complete the following steps.

Procedure

1. Open the New Application wizard by using one of the following methods:
   - Click **File** > **New** > **Application**.
   - In the Application Development view, click **New**, then click **Start by creating an application**.
   - Right-click the white space of the Application Development view, then click **New** > **Application**.
2. Enter a name for the application, then click **Next**.
3. Existing libraries are listed; select the libraries that you want to add to the application.

   If libraries are referenced by the libraries that you have selected, they are shown.
4. Click **Finish**.

   An application is created and is shown in the Application Development view. You can expand the application to view library references. Shared libraries are stored in a Reference Libraries folder, and static libraries are stored in an Included Libraries folder.

What to do next

- If you have not already done so, you can create a library by following the instructions in "Creating a library" on page 1964.
- To create resources in an application, see "Creating resources in an application" on page 1964.

- To add and remove project references to the application, see "Referencing resources in other libraries" on page 1976.
- To include an existing project in an application, see "Adding a project to an application, integration service, or library" on page 1979.

Creating resources in an application

You can create resources such as message flows, maps, and ESQL files in an application.

Before you begin

Learn more about applications by reading "Applications" on page 1943.

This topic assumes that you have created an application. For more details, see "Creating an application" on page 1963.

About this task

An application is a specialized container in which you create and maintain all the resources associated with a business solution. You can create resources such as message flows, maps, broker schemas, and database definitions. You can also generate message models and import resources such as CORBA IDL files, XSLT files, and database records.

To create resources in your application, complete the following steps.

Procedure

1. Click **File** > **New**.

   (Alternatively, in the Application Development view, right-click an application, then click **New**.)
2. Select the resource to create.

   The appropriate wizard opens; for example, the **New Message Flow** wizard.
3. If the application name is not already provided, select it from the list of available applications.
4. Follow the instructions in the wizard, then click **Finish**.

   The new resource is displayed under the application in the Application Development view.

Creating a library

Create a library to contain a logical grouping of related code, data, or both, that can be reused.

About this task

The order in which you create resources is flexible. You can begin by creating an application and adding resources to it, or you can begin by creating libraries of resources, then create an application that refers to them. Libraries are optional; create one if you want to reuse resources.

Two types of library exist in IBM App Connect Enterprise: *shared libraries* and *static libraries*. A shared library can be deployed directly to an integration server or in a BAR file with referencing applications. If that shared library is updated, the changes are immediately available to all applications or shared libraries that refer to it.

A static library is deployed with the application that references it. Each application that references that static library is deployed with its own private copy of that library. If a static library is updated, each application that references it must be repackaged and redeployed with the updated static library.

A static library is the original container that was introduced in WebSphere Message Broker Version 8.0. In IBM Integration Bus 10.0 and later, the preferred way to share resources is to store them in a shared library. If you want each application to use a different version of the contained resource, you can use a static library.

For more information about libraries, see “Libraries” on page 1945.

Procedure

To create a library, complete the following steps.

1. Open the **New Library** wizard by using one of the following methods in the IBM App Connect Enterprise Toolkit:
   - Click **File** > **New** > **Library**.
   - In the Application Development view, click **New**, then click **Start by creating a library**.
   - Right-click the white space in the Application Development view, then click **New** > **Library**.
2. Enter a name for the library.
3. Select either **Shared Library** or **Static Library**.
4. If this library is the first one to be created, click **Finish**. If another library exists, you can click **Next** to add a reference to an existing library.
5. Optional: If you click **Next**, existing libraries are listed. Select the existing libraries that you want to reference from the new library.

   A shared library can reference other shared libraries only; a shared library cannot reference a static library. Similarly, a static library can reference other static libraries only. If you add a reference to a static library that refers to other static libraries, all referenced static libraries are included.
6. Click **Finish**.

Results

A library is created and is shown in the Application Development view. You can expand the library to view library references. Shared libraries are represented by the shared library icon  . Static libraries are represented by the static library icon  . A shared library that is referenced by an application or another shared library is shown in the Referenced Libraries folder for that application or shared library. A static library that is referenced by an application or another static library is shown in the Included Libraries folder for that application or static library.

What to do next

- Create an application by following the instructions in “Creating an application” on page 1963.
- To create resources in a library, see “Creating resources in a library” on page 1965.
- To add and remove project references in the library, see “Referencing resources in other libraries” on page 1976.

Creating resources in a library

You can create resources in a library that you want to reuse in multiple solutions.

Before you begin

- Find out more about libraries in “Libraries” on page 1945.
- You can add resources to an existing library. (For more information, see “Creating a library” on page 1964.) Alternatively, you can create a library when you create a resource.

About this task

A library is a container for resources that you want to reuse for multiple solutions. You can create resources such as maps, broker schemas, and database definitions. You can also generate message models and import resources such as CORBA IDL files, SCA files, and XSLT files. You can create subflows in a shared library, but you cannot create message flows in a shared library.

You can create resources in an existing library, or you can create a library when you create a resource.

Creating resources in an existing library

Procedure

1. In the Application Development view, right-click a library, then click **New**.
2. Select the resource to create.

   When you select a resource to create, the appropriate wizard opens; for example, the **New Message Model** wizard.
3. If the library name is not already provided in the **Application or Library** field, select it from the list of available libraries.
4. Follow the instructions in the wizard, then click **Finish**.

   The new resource is displayed under the library in the Application Development view.

Creating a library when you create a resource

Procedure

1. In the Application Development view, click **New**, or right-click white space, then click **New**.
2. Select the resource to create.

   When you select a resource to create, the appropriate wizard opens; for example, the **New Message Model** wizard.
3. Create a library by clicking **New** next to the **Application or Library** field.
4. Select **Library**, then click **OK**.
5. Enter a name for the library, then click **Finish**.

   A new shared library is created and shown in the Application Development view.
6. Follow the instructions in the wizard, then click **Finish**.

   The new resource is displayed under the library in the Application Development view.

Creating a broker schema

To organize your resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create a message flow, but you can create additional schemas.

Before you begin

The order in which you create your resources is flexible. You can add a schema to an existing application, library, or integration project, or you can create a container when you create the schema. To create a container first, follow the instructions in the following topics:

- “Creating an application” on page 1963
- “Creating a library” on page 1964
- “Creating an integration project” on page 1969

If your schema file is intended for use by multiple solutions, consider creating it in a shared library.

To find out more about schemas, see “Broker schemas” on page 498.

About this task

When you create a message flow, a schema is created by default. The default schema is not displayed in the Application Development view. Only additional schemas that you create are displayed in the Application Development view, under the containing project. For more information about creating a message flow, see “Creating a message flow” on page 574.

To create additional schemas, complete the following steps.

Procedure

1. To open the **New Broker Schema** wizard, use one of the following methods.
   - Right-click the space in the Application Development view and click **New** > **Broker Schema**.
   - Click **File** > **New** > **Other** > **IBM App Connect Enterprise - Application Development** > **Broker Schema**, then click **Next**.
   - Right-click an application, static library, or integration project, then click **New** > **Broker Schema**.
2. Select the name of the application, library, or integration project in which to create the schema.

   Alternatively, create a new container by clicking **New**.
3. Enter a name for the schema.

   Choose a name that reflects the resources that the schema contains; for example, if you want to use this schema for message flows for retail applications, you might give it the name `Retail`.

   A schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore symbol (_). You can use the period to provide a structure to the name, for example `Stock.Common`.
4. Click **Finish**.

   The schema is shown in the Application Development view under the relevant container.

   If category view is selected in the Application Development view when you create the schema, you see a message to say that the schema was created. However, it might not be visible in the Application Development view when it is empty. To show the new schema in the Application Development view, click **Hide Categories**  on the Application Development view toolbar.

Results

The schema directory is created in the project directory. If the schema name is structured by using periods, further subdirectories are defined. For example, the schema `Stock.Common` results in a directory `Common`, in directory `Stock`, in the application, library, or integration project directory.

Creating a solution based on WSDL or XSD files

You can use existing WSDL or XSD files as the basis for your solution.

Procedure

1. In the Application Development view of the IBM App Connect Enterprise Toolkit, click **New**, then click **Start from WSDL and/or XSD files**.
2. Decide how you want the resources to be organized by creating an application, library, or integration service to contain them, then click **Next**.

   An application typically contains all the resources required for your solution. A shared library typically contains resources that you might reuse for other solutions. You can choose to contain all of your resources in an application if you do not plan to reuse them elsewhere. Alternatively, you can choose to store all your resources (except for the main message flow) in a shared library for reuse.

   Depending on the option that you select, default names are provided for the containers that are created, but you can edit the names.

3. Select the WSDL or XSD files that you want to use, then click **Next**.
   - To choose WSDL or XSD files that exist in your workspace, select **Use resources from the workspace**.

     Your resources are shown in the same format as the Application Development view. Expand the folders and select the resources that you want to use. Resources might be filtered if you have selected a working set.
   - To choose WSDL or XSD files that exist outside your workspace, select **Use external resources**.

     To locate your resources, enter a directory name or click **Browse**.

   If the only use of the XSD file is from the WSDL bindings, you do not have to select an XSD file on which a selected WSDL files depends.
4. Select one or more bindings for each of the WSDL files that you selected, then click **Finish**.

Results

Your container and resources are created and are shown in the Application Development view.

What to do next

After you import a WSDL file into a container, drag the WSDL file onto the Message Flow editor to create a message flow. You need to open or create a message flow. If you are dragging a WSDL file from a shared library into a message flow in an application or other container, that container must reference the shared library. When you drag a WSDL file onto a message flow, the **Configure New Web Service Usage** wizard guides you through the creation of the message flow.

Creating a solution based on an existing message set

Import or copy an existing message set into your workspace, then create an integration project and message flow that refer to that message set.

About this task

To create a solution that is based on an existing message set, complete the following steps.

Procedure

1. Click **File** > **New** > **Other**, or right-click anywhere in the Application Development view and click **New** > **Other**.

   The New Project wizard opens.
2. Expand **Integration Bus - Message Set Development**.
3. Click **Start from existing message set**, then click **Next**.
4. Set up the basic resources that are required to develop an application from an existing message set, then click **Next**.
   - If the message set that you want to use is in a compressed (`.zip`) file, select **Import a message set from a .zip file**, and either type the location of the message set in the **.zip file** and **Compressed message set** fields, or click **Browse** and select, and open, the `.zip` file from the list that is displayed. Then select the required message set.

     If the `.zip` file that you specify does not contain a message set, you receive a message. You can then type a different location for the message set in the **.zip file** field. Otherwise, click **Cancel**.
   - If the message set that you want to use is not in a compressed file, click **Create a new message set by copying an existing message set** and type into the **Message set to copy** field the name of the message set file that you want to copy.

     A list is displayed of the message set names from which you can choose. Message sets are filtered to show resources in the active working set only.

5. If necessary, you can change the default name for the integration project that is created.

   By default, a message flow and working set are also created. You can choose different names for these resources. If the message set is copied from a `.zip` file that is a project interchange file, you cannot edit the names of the message set project and the message set. In this case, the names are imported from the `.zip` file.
6. Click **Finish**.

   The new message set project, message set, integration project, and message flow are created. The integration project contains a project reference to the message set project. A new working set is also created, if required. The new projects are displayed in the specified working set. The contents of the message set project and the integration project are displayed in the Application Development view. The message flow is opened in the message flow editor.

What to do next

The next step is to develop the message flow. For more information about how to start, see "Designing a message flow" on page 583.

Creating an integration project

Create an integration project to contain resources such as message flows, maps, and database definitions.

Before you begin

Read the concept information in "Integration projects" on page 1959.

About this task

Integration projects replaced message flow projects in IBM Integration Bus 10.0.

If you are not ready to create an application or library yet, you can use an integration project to hold resources in the meantime. When you are ready, you can either create an application or library and move the resources from the integration project, or you can convert your integration project to an application or library.

You can create an integration project when you create resources, such as message flows and maps.

To create and populate an integration project manually, complete the following steps.

Procedure

1. Open the **New Integration Project** wizard by clicking **File** > **New** > **Integration Project**.
2. Enter a name for the project and select any existing projects to which you want to refer. Click **Next**.
3. Select any existing static libraries to which you want to refer, then click **Finish.**

Results

The new integration project is created and shown in the Independent Resources folder of the Application Development view.

What to do next

When you have created an integration project, you might want to create resources to add to the project. For more information, see "Creating resources in an integration project" on page 1970.

To see the project references for your integration project, right-click the integration project and click **Properties** > **Project References**. The projects that your integration project references are selected.

To convert an integration project to an application or library, see “Converting existing projects to applications and libraries” on page 1970.

Creating resources in an integration project

You can create resources such as message flows, maps, and ESQL files in an integration project.

Before you begin

Learn more about integration projects by reading “Integration projects” on page 1959.

This topic assumes that you have created an integration project. For more details, see “Creating an integration project” on page 1969.

About this task

An integration project is a specialized container in which you create and maintain all the resources that are associated with one or more message flows. To create resources in your integration project, complete the following steps.

Procedure

1. Click **File** > **New**.

   (Alternatively, expand the Independent Resources folder of the Application Development view, right-click the integration project, then click **New**.)
2. Select the resource to create.

   The appropriate wizard opens; for example, the **New Message Flow** wizard.
3. Follow the instructions in the wizard, then click **Finish**.

   The new resource is displayed in the Application Development view, under the integration project.

What to do next

You can also reference resources that exist outside the integration project (such as a Java project or Data Design project) by adding project references. Right-click the integration project, then click **Properties** > **Project References**. Select the projects that you want to reference in your integration project.

Converting existing projects to applications and libraries

Two options are available to convert existing projects to applications or libraries. Use **Convert Single Project** to convert a project with no references. Use **Analyse and Convert Multiple Projects** to convert one or more projects and any referenced resources.

Before you begin

- Read the concept information about applications, libraries, and integration projects in “Resource management overview” on page 1942.
- To convert projects from previous versions of IBM App Connect Enterprise, you must first import them. For instructions, see “Migrating development resources to IBM App Connect Enterprise 12.0” on page 62.

About this task

You can convert an integration project, Java project, or message set project into an application or static library. For example, you might want to convert a project in the following situations:

- You imported an existing project from a previous version of IBM App Connect Enterprise. You want to use that project as the basis for a new application. You can convert the project into an application, then add the rest of the resources that are required to complete your solution.

- You have an existing project that contains resources that you want to reuse in several solutions. You can convert the project into a static library, add any other required resources, then refer to that library from the appropriate applications.
- You imported your resources from a previous version of IBM App Connect Enterprise, and your message flow projects were migrated to integration projects. You can convert all or some of your projects (such as integration projects, Java projects, and message set projects) automatically to applications and static libraries.

To convert a project to an application or library, follow the appropriate instructions.

Procedure

- To convert a project that has no references to other resources, a simple conversion to an application or static library is sufficient; see "Converting a single project to an application or library" on page 1971.
- To convert all imported projects, or a project that contains references to other resources, a conversion wizard is provided, which uses rules to decide how to convert your resources. The wizard also converts associated projects, such as Java projects and message set projects, that are referenced by the selected projects; see "Converting one or more projects with the conversion wizard" on page 1972.
- By using the conversion wizards, you can convert a project to a static library. You can then convert a static library to a shared library manually; see "Converting a static library to a shared library" on page 1974.

Converting a single project to an application or library

Convert a project with no references to other resources by using the **Convert to Application or Library** wizard.

About this task

The following restrictions apply when you convert a single project to an application or library:

- You can convert only integration projects, Java projects, and message set projects.
- You can select multiple projects to convert to applications or static libraries, but you cannot convert more than one integration project at a time. To convert multiple integration projects, you must use the **Analyse and Convert Multiple Projects** wizard.
- You cannot convert a project that refers to a library.
- If an integration project is referenced by another integration project, you can convert it to a static library only.
- When you select the **Convert single project** option, no precondition checks are made, such as checking for references to projects that do not exist. If your project might contain references to other resources, use the **Analyse and Convert Multiple Projects** wizard to analyze all projects and resources. To complete precondition checks before you convert a project, select the option **Analyse and Convert Multiple Projects**.

To convert a single project to an application or library, without making precondition checks, complete the following steps.

Procedure

1. In the Application Development view, right-click the project or projects that you want to convert, then click **Convert to application or library** > **Convert single project**.
2. Select either **an application** or **a library**.
3. Click **Finish**.

Results

The project is converted into an application or static library and is shown in the Application Development view. The contents of the original project are now shown under the new application or library.

What to do next

After you create an application or library, you can continue with any of the following tasks:

- "Creating resources in an application" on page 1964
- "Creating resources in a library" on page 1965
- "Referencing resources in other libraries" on page 1976
- "Deployment rules and guidelines" on page 2465

If you converted an integration project that referred to a Java project, you can now convert the Java project to a library so that it can be used by other applications. In the Application Development view, right-click the Java project, click **Add project to existing container**, then select an existing shared library or other container.

You can also convert an application to a static library, or a shared or static library to an application, by following the instructions in "Converting between applications and libraries" on page 1974.

Converting one or more projects with the conversion wizard

Convert all imported projects, or a project that contains references to other resources, by using the **Convert Projects to Applications and Libraries** wizard.

About this task

To convert projects to applications or static libraries, including precondition checks, complete the following steps.

Procedure

1. In the Application Development view, select the resources that you want to convert.

   You can select multiple projects in the Independent Resources folder, or you can select all projects by right-clicking the Independent Resources folder.
2. Right-click the Independent Resources folder, or one of the selected projects, and click **Convert to application or library** > **Analyze and convert multiple projects**.

   The **Convert Projects to Applications and Libraries** wizard opens.
3. Projects that you can convert are listed. Projects that you selected in step 2 are preselected in the wizard. Select the projects to convert, then click **Next**.

   The wizard completes a precondition check. If any errors are detected that would prevent successful conversion, those errors are listed. For example, an integration project cannot be converted if it references a project that does not exist. You cannot continue with the conversion until all errors are fixed.
4. Select an error and click **Fix selected**.

   Alternatively, you can fix all the errors that are listed by clicking **Fix all**. If more than one solution is available for an error, you can select the solution from the Quick Fix dialog box that opens. An error that was fixed is indicated by a green tick.

   After you apply one or more fixes, if you change your mind, you can undo fixes. Click **Cancel**, then specify whether the fixes made during the precondition check are to be undone.

   After you apply fixes to the errors, the wizard runs the precondition check again. If new errors occur, they are listed so that you can fix them.

   When no errors exist, click **Next**.

5. Review the proposed conversion actions.

   The wizard analyzes the required conversion actions and shows a summary of changes that it will make. The proposed new applications and libraries are shown in tree form. Expand an application or library to see the resources that it references. The Conversion Actions pane lists the changes that will be made to convert the selected resource.

   The wizard applies the following rules to convert projects to applications or libraries:

   - A top-level integration project, or an integration project that is not referenced by another integration project, is converted to an application.
   - An integration project that is referenced by another integration project is converted to a static library.
   - If you choose to convert a referenced project that is not an integration project, it is converted to a static library, except for data design projects and adapter connection projects. These projects are unchanged and are referenced by the application or library. If you choose not to convert the referenced project, it remains unchanged, and the referencing project refers to the original project.

     For example, an integration project refers to a message set. If the integration project and the message set project are both selected for conversion, the integration project is converted to an application, and the message set project is converted to a static library. If, however, the integration project is selected for conversion, but the referenced message set project is not selected, the integration project is converted to an application that refers to the message set. The message set is not converted.
   - A project that is not an integration project, and that is referenced by multiple projects, is converted to a wrapper library. A wrapper library is a library that refers to the original project, such as a message set project. Project references from other projects are updated to refer to the wrapper library.
   - A stand-alone project that is not an integration project is converted to a static library.
   - An integration project that contains broken references is not converted. The broken references are reported so that you can fix them.
6. The conversion changes cannot be reversed. Therefore, you are prompted to back up your workspace by clicking **Export**.
7. Optional: For future reference, you can save the conversion actions for all converted projects by clicking **Save all conversion actions to file**. You can then save the single document as a `.txt` file.
8. To complete the proposed changes, click **Finish**.

Results

Your selected projects, and affected referenced projects, are converted to applications and static libraries and are shown in the Application Development view.

What to do next

After you create an application or library, you can continue with any of the following tasks:

- “Creating resources in an application” on page 1964
- “Creating resources in a library” on page 1965
- “Referencing resources in other libraries” on page 1976
- “Deployment rules and guidelines” on page 2465

If you converted an integration project that referred to a Java project, you can now convert the Java project to a library so that it can be used by other applications. In the Application Development view, right-click the Java project, click **Add project to existing container**, then select an existing shared library or other container.

You can also convert an application to a static library, or a shared or static library to an application, by following the instructions in “Converting between applications and libraries” on page 1974.

Converting a static library to a shared library

When you use the **Convert to Application or Library** wizards, you can convert one or more projects to static libraries. You can then convert a static library to a shared library manually

Before you begin

Before you convert a static library to a shared library, check that the resources in your static library are supported in a shared library. For more information, see "Resources that are not supported in a shared library" on page 1949.

About this task

When you convert a project or application to a library, that library is a static library. To convert a static library to a shared library, complete the following steps.

Procedure

1. Create a shared library in the IBM App Connect Enterprise Toolkit by clicking **File** > **New** > **Library**.
2. Enter a temporary name for the shared library, then click **Finish**.

   The name must be unique.
3. Move the contents of the static library to the shared library.

   You must preserve the folder structure when you move the contents from a static library to a shared library. To reveal all the folders that you must move, click **Hide Categories**  on the Application Development view toolbar.

   A message warns that moving resources to another location breaks project references. Accept this message. You will fix the broken references by renaming the libraries.
4. Rename the static library so that you can use the old name for the new shared library.

   Right-click the static library, click **Rename**, and enter a name.
5. Rename the shared library so that it has the old name of the static library.

   By renaming the shared library, broken project references are fixed.
6. After you have tested the shared library, delete the static library.

Results

The new shared library is shown in the Application Development view with the appropriate icon for a shared library (). If the shared library is referenced by another resource, it is shown in the Referenced Libraries folder for that resource.

Converting between applications and libraries

You can convert an application into a static library, or you can convert a static or shared library into an application.

Before you begin

- Read the concept information about applications, libraries, and integration projects in "Resource management overview" on page 1942.
- To convert projects from previous versions of IBM App Connect Enterprise, you must first import them. For instructions, see "Migrating development resources to IBM App Connect Enterprise 12.0" on page 62.
- To convert a project to an application or library, see "Converting existing projects to applications and libraries" on page 1970.

About this task

Applications are designed to contain all the resources that you need for a particular solution. Your main message flows are typically contained by an application. Libraries are designed to contain objects for reuse. Libraries typically contain reusable helper routines and resources such as subflows, ESQL modules, message definitions, and Java utilities. You might want to convert an application to a static library if it contains resources that you want to reuse for other solutions. You might want to convert a library to an application if it contains all the resources that are required to build your solution. If your application contains runnable message flows, a warning is displayed if you try to convert it to a library.

To convert an application or library to a different project type, complete the following steps.

Procedure

1. In the Application Development view, right-click the application or library that you want to convert, then click **Convert to Application** or **Convert to Library**.

   If you are converting a library to an application, conversion happens immediately. If necessary, you can rename the converted application by right-clicking it and selecting **Rename**.
2. If you are converting an application to a static library, and your application contains runnable flows, read the warning message. To leave the application as an application, click **Cancel**. If you still want to convert the application to a library, click **Convert**.

   The application is converted to a static library. To share the resources in this library between multiple solutions, you can convert the static library to a shared library (see "Converting a static library to a shared library" on page 1974).

Results

The converted application or library is displayed with its resources in the Application Development view.

Referencing resources from a container

To include resources in your application, library, or integration service, add references to libraries, or include projects in the container.

About this task

An application contains all the resources that you need to build a business solution. An application can use the resources that are stored directly in that application, and it can use resources that are stored in other libraries or projects. Resources in the IBM App Connect Enterprise Toolkit exist in specific types of project. Resources can refer to each other; for example, a message flow can refer to subflows or maps. Referenced resources can be in the same project, or in another referenced project. Applications, libraries, and integration services can refer to the resources in other libraries. Applications and libraries can also refer to resources in other projects, such as Java projects or data design projects.

For instructions about how to refer to resources in libraries, or include projects in containers, see the following topics:

- "Referencing resources in other libraries" on page 1976
- "Adding a project to an application, integration service, or library" on page 1979
- "Removing a project from a container" on page 1981

Referencing resources in other libraries

To include resources in your application, library, or integration service from other libraries, add references to those libraries.

About this task

The best way to make a resource available to multiple applications, libraries, or integration services is to store it in a shared library. For example, you might have some common error-handling code that you want to make available to multiple applications. You can store those common files in a shared library, then you can add a reference from your application to that shared library. A shared library can also reference another shared library. A shared library is deployed with or before the container that refers to it. If you update the contents of that library and redeploy it, those changes are available to all containers that refer to the library.

For examples of how to use shared libraries to share resources between multiple solutions, see the following topics:

- "Example: Sharing Java classes between solutions" on page 1976
- "Example: Sharing map files between solutions" on page 1978

If you reference a static library from an application, that static library is embedded in the application at deployment time. The library is effectively private and is not accessible to other applications or message flows outside the referring application.

A static library can reference another static library only. Similarly, a shared library can reference another shared library only.

The following steps describe how to add library references.

Procedure

1. In the Application Development view, right-click an application, library, or integration service and click **Manage library references**.
2. Select the libraries that you require and click **OK**.

   If the libraries that you select refer to other libraries, those dependent libraries are shown.
3. To check which resources will be deployed with your solution, look at the Prepare tab of the BAR file editor. When you expand the application, integration service, or library, library references are listed.

What to do next

To remove a reference to a library, right-click an application, library, or integration service and click **Manage library references**. Clear the check box next to the library references that you want to remove and click **OK**.

Example: Sharing Java classes between solutions

When you need to use the same Java classes in multiple solutions, store those Java classes in a shared library. You can access the Java classes in the shared library by using JavaCompute nodes in your applications.

About this task

If you update those Java classes and redeploy the shared library, those changes are automatically available to all applications that reference that shared library. You do not need to redeploy the referencing applications.

Java classes that are deployed in a shared library are not available to the integration server-wide class loader. When you deploy a shared library that contains Java files, a new class loader is created for that shared library. This class loader contains all Java classes in the shared library, as well as the Java classes

from all referenced shared libraries. Java classes in shared library class loaders are isolated from Java classes in the integration server-wide class loader.

A Java class can exist in multiple class loaders if that Java class is contained in a shared library that is referenced by other shared libraries. When you update that Java class by redeploying the shared library that contains it, all class loaders for all shared libraries that contain that Java class are deleted and re-created.

If you include a JavaCompute node in a subflow in a shared library, that node can access the Java classes in that shared library and any referenced shared libraries. If you include a JavaCompute node in a flow or subflow in an application, that node can access the Java classes in a shared library and any shared libraries that are referenced by that shared library. In this case, you must specify a shared library qualifier in the form `{shlib1}` as the Java classloader service. Java classes in applications, static libraries, or independent projects cannot access Java classes in shared libraries.

The order in which you create your resources is flexible. The following steps describe one way to share Java classes between multiple applications.

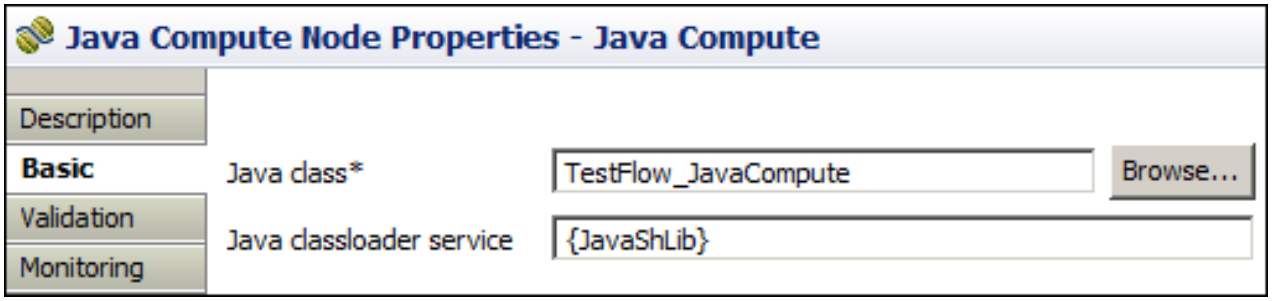
Procedure

1. In the IBM App Connect Enterprise Toolkit, create a shared library.
2. If you have existing Java classes, add them to the shared library. Otherwise, you can create your Java classes when you add a JavaCompute node to a message flow.
3. Create one or more applications to contain the message flows and other resources for your solutions.
4. Ensure that your applications refer to the shared library that contains the Java classes.
5. Create one or more message flows in your applications that contain JavaCompute nodes.
6. Configure each JavaCompute node to point to the Java classes in the shared library.

   Double-click the JavaCompute node and set the **Source folder** field to the shared library that contains the Java classes.
7. If you have existing Java classes, click **Finish**.
8. If you have not yet created your Java classes, click **Next** and select a template for your class. Define the Java build settings, then click **Finish**.

Results

The Java class file opens in the editor. If you view the basic properties for the JavaCompute node, the **Java class** field is populated with the location of the specified Java class. The **Java classloader service** field contains a qualifier for the shared library that contains the Java class in the form {*sharedLibraryName*}.



What to do next

After you create all the resources that your solution requires, deploy the shared library and the referencing applications.

You can deploy your resources by dragging the shared library and the applications onto the integration server. If you use this method, you must deploy the shared library before the applications that reference it. Alternatively, you can add the applications and shared library to a BAR file, then deploy that BAR file.

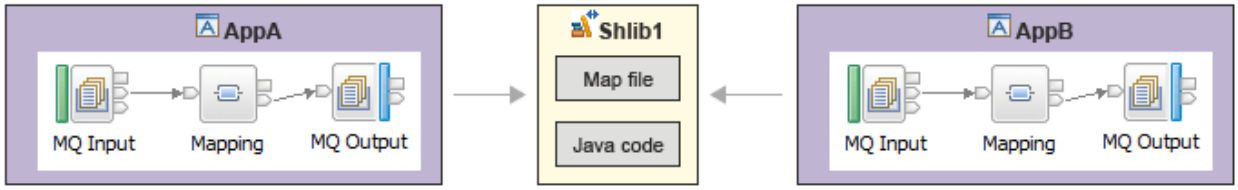
Example: Sharing map files between solutions

When you need to use the same map files in multiple solutions, store those map files in a shared library. You can access the map files in the shared library by using Mapping nodes in your applications.

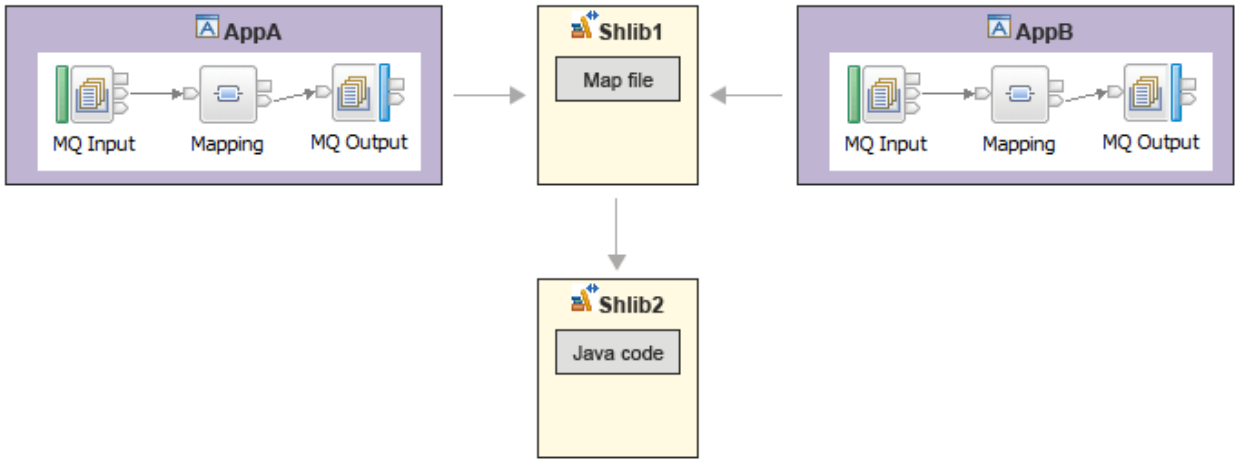
About this task

In this scenario, a map is used by multiple solutions. Therefore, the map is contained in a shared library so that multiple applications can reference it. This map uses a custom Java transformation, which is also used by multiple maps. Therefore, the Java code is also stored in a shared library, so that multiple shared libraries can reference it.

Example 1: The Mapping nodes in two different applications use the map file in a shared library. That map file uses a custom Java transformation, which is stored in the same shared library.



Example 2: The Mapping nodes in two different applications use the map file in a shared library. That map file uses a custom Java transformation, which is stored in another referenced shared library.



To create the scenario shown in example 2, complete the following steps. These steps assume that the Java code already exists in a shared library.

Procedure

1. In the IBM App Connect Enterprise Toolkit, create a message flow in an application and add a Mapping node.
2. Double-click the Mapping node. The **New Message Map** wizard opens.
3. In the **Container** field, click **New**.
4. Specify a name for the library, then click **Next**.
5. Select the shared library that contains the Java code that is required by the map, then click **Finish**.
6. To specify an alternative location for the mapping schema files, clear **Use default broker schema** and select the location of the mapping schema files. Click **Next**.
7. Select the mapping input and output models, then click **Next**.
8. The output domain is set automatically according to the selected output model. Click **Finish**.
9. Save the message flow.

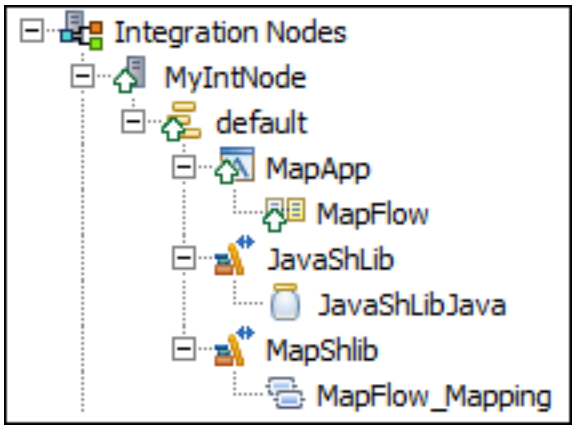
Results

You can view your resources in the Application Development view. An application contains the message flow. One shared library contains the message map, and another shared library contains the Java code that is required by that message map.

If you view the properties for the Mapping node, you can see that the **Mapping routine** field is set to the location of the map file. The location is in the following format: `{default}:MapFlow_Mapping in Shared Library MapShlib`. This value specifies that the map file is called `MapFlow_Mapping`, it uses the default broker schema, and it is contained in the shared library `MapShlib`.

What to do next

Deploy your resources. Shared libraries that are referenced by applications must be deployed with or before the applications that refer to them. You can deploy the shared libraries directly to the integration server before you deploy the applications. Alternatively, you can include the applications and shared libraries in a BAR file and deploy the BAR file.



If you update the map file or Java code in the shared libraries, those changes are available automatically to all applications that refer to those shared libraries.

Adding a project to an application, integration service, or library

If a project contains resources that are required for your solution, you can add that project to an application, integration service, or library.

Before you begin

- Read the concept information about applications, integration services, and libraries in "Resource management overview" on page 1942.
- To include projects from previous versions of IBM App Connect Enterprise, you must first import them.
- As an alternative to including a project in an application or library, you can convert a project to an application or library.

About this task

You can include the projects, such as Java projects, in an application, integration service, or library.

The following resources are not supported in shared libraries:

- WebSphere Adapters files
- Decision services
- IDL files
- Message flows

- Message sets
- Static libraries
- XSL files

You can add a project in an existing container in two ways:

- Begin with a project and choose which application or library to include it in.
- Begin with an application or library and choose which projects to include in it.

The following sections describe these methods.

Beginning with a project

Procedure

1. In the Application Development view, expand the Independent Resources folder and locate the relevant project.
2. Right-click the project and click **Add project to existing container**.

   A wizard of the same name opens. The wizard also lists any projects that are referenced by your selected project.
3. Select the application, integration service, or library in which you want to include the project, then click **OK**.

Beginning with an existing container

Procedure

1. In the Application Development view, right-click an application, integration service, or library, then click **Manage included projects**.

   A wizard lists all the projects that you can add to your application or library. The wizard does not list projects that are already included in other applications or libraries. Projects that refer to a project that is included in a different application or library are also not listed. If the project that you select refers to other projects, those projects are listed in the wizard for your information.
2. Select the project or projects that you want to include in the application or library, then click **OK**.
3. Eclipse also provides a method of managing project references. Right-click an application, library, or project in the Application Development view and click **Properties**, then **Project References**. However, use this method with caution. A list of all applications, libraries, and projects are listed, regardless of whether they can be referenced by the selected application, library, or project.
   For example, if you use this method to manage project references for a library, the interface allows you to select an application as a reference. However, a library cannot reference an application. Use this method only in the following circumstances:

   - If you want to refer to a project like a Java project or data design project from an integration project.
   - You can also use project references to enable content assist in the ESQL editor. (Content assist is context-sensitive help that displays valid ways in which a code statement can be completed.) If you set up a project reference from an integration project that contains ESQL code to a message model schema file, the ESQL editor can display a list of valid message references.

Results

The container now includes a reference to your selected project or projects. Those projects are no longer visible under the Independent Resources folder in the Application Development view, but are listed under the application, integration service, or library. To check which resources will be deployed with your solution, look at the Prepare tab of the BAR file editor. When you expand the application, integration service, or library, included projects are listed.

What to do next

To remove a project from a container, see "Removing a project from a container" on page 1981.

Removing a project from a container

If you solution no longer needs the resources in a project, you can remove that project from an application, integration service, or library.

Before you begin

- Read the concept information about applications, integration services, libraries, and integration projects in "Resource management overview" on page 1942.

This topic assumes that you have already included a project in an application or library. For more information, see "Adding a project to an application, integration service, or library" on page 1979.

About this task

You can remove the following types of project from an application or library:

- Java project
- Data design project
- Message set project

Procedure

To remove a project from a container, complete the following steps.

1. In the Application Development view, right-click an application or library, then click **Manage included projects**.

   A wizard lists all the projects that are in your application or library as well as all the projects that you can add to your application or library. The projects that are currently in the container are selected. The wizard does not list projects that are already included in other applications or libraries. Projects that refer to a project that is included in a different application or library are also not listed. If the project that you select refers to other projects, those projects are listed in the wizard for your information.
2. Clear the check boxes for the projects that you want to remove from the container, then click **OK**.

Results

The projects are removed from the application, integration service, or library. Those projects are no longer visible under the application, integration service, or library, but are listed under the Independent Resources folder.

Focusing on an application or library

Focus on an application or library to show only the resources that are associated with that container. You can also filter projects by using working sets.

Before you begin

This topic assumes that you have already created an application or library, as described in the following topics:

- "Creating an application" on page 1963
- "Creating a library" on page 1964

For more information about working sets, see Working sets.

About this task

If you focus on an application or library, the Application Development view shows only the selected container and its contents.

To focus on an application or library, complete the following steps.

Procedure

1. In the Application Development view, right-click an application or library, then click **Focus on Application** or **Focus on Library**.
2. To show all resources in the Application Development view, right-click the application or library, then click **Focus on Application** or **Focus on Library** again to deselect the option.

Deleting an integration project

An integration project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed in the project in the Application Development view. If you do not want to retain an integration project, you can delete it.

Before you begin

- The instructions in this topic assume that you have already created an integration project, as described in "Creating an integration project" on page 1969.
- Read the concept topic about "Integration projects" on page 1959

About this task

When you delete an integration project in the IBM App Connect Enterprise Toolkit, you can choose to delete the project and its resources. If you are using a shared repository, the repository might retain a copy of a deleted resource.

To delete an integration project, complete the following steps.

Procedure

1. In the Application Development view, select the integration project that you want to delete, and click **Edit** > **Delete**.

   You can also press Delete, or right-click the project and click **Delete**.
2. To delete the contents of the integration project, select **Also delete contents under *directory_name***.

   If you choose not to delete the contents of the integration project directory, all the files and the directory itself are retained. If you later create another project with the same name, and specify the same location for the project (or accept the location as the default value), you can access the files previously created.

   If you choose to delete all the contents, all files and the directory itself are deleted.
3. Click **Yes** to delete the integration project and selected resources.

Results

If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

If you are using the local drive or a shared drive to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.

Deleting a schema

You can delete a broker schema that you have created in an application or library if you no longer need it.

Before you begin

This topic assumes that you have already created a broker schema, as described in "Creating a broker schema" on page 1966. For more information about schemas, see "Broker schemas" on page 498.

About this task

The Application Development view is populated with all the applications and libraries to which you have access. A broker schema is contained in one of these containers.

To delete a broker schema, complete the following steps.

Procedure

1. In the Application Development view, expand the appropriate application or library.
2. If the schema contains resources, delete them.

   To delete the schema, it must be empty. The schema might not be visible in the Application Development view when it is empty. To show the new schema in the Application Development view, click **Hide Categories**  on the Application Development view toolbar.
3. Select the broker schema, then click **Edit** > **Delete**.

   A confirmation dialog box is displayed.
4. Click **Yes** to delete the broker schema directory or **No** to cancel the delete request.

   When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be careful to select the correct resource when you complete this task.

Results

Exporting XSD schema files from an application or library

Export XSD schema (`.xsd`) files from an application or library for use in external tools.

About this task

You can export `.xsd` files without annotations specific to WebSphere Adapters from an application or library, so that you can use them in other programs.

Procedure

1. In the Application Development view:
   - Right-click your application and click **Export XSDs from Application**. The **Export XSDs from Application** window opens.
   - Right-click your library and click **Export XSDs from Library**. The **Export XSDs from Library** window opens.
2. Enter the directory that you want to export the `.xsd` files to, and click **OK**.

Results

Your `.xsd` files are exported from the application or library to the directory you selected.

Developing integration solutions by using integration services

To create web services solutions, create integration solutions by using integration services.

Before you begin

If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see "Developing message flows" on page 482 and "Constructing message models" on page 2133.

About this task

In IBM App Connect Enterprise, an integration service is a specialized application that has a tightly defined interface and operations. You can configure the integration service so that you can call the operations from a web service (by using a SOAP/HTTP binding), or from a JavaScript program (by using a JavaScript API binding). For more information, see "Integration services" on page 1944.

Procedure

To build an integration service:

1. Create an integration service interface.

   You have the following options:

   - Create an integration service interface from scratch. For more information, see "Creating an integration service from scratch" on page 1986.
   - Create an integration service interface by importing an existing WSDL. For more information, see "Creating an integration service based on a WSDL file" on page 1987.
2. If your integration service is not based on imported WSDL or a Business Process Manager service, specify the interface by defining the operations. See "Defining the operations in a service interface" on page 1989.
3. Optional: Create a library to contain a logical grouping of related code, schemas, message models, and other resources that can be reused. For more information, see "Creating a library" on page 1964.
4. Optional: Define the required library references in the integration services, as described in "Referencing resources in other libraries" on page 1976. To include one or more libraries of resources in your integration service, you create references to those libraries.
5. Optional: Create other resources that you need in a library. These resources can be reused by other integration services or applications. For more information, see "Creating resources in a library" on page 1965.
6. Use the Service editor to implement each service operation as a subflow. See "Implementing an integration service operation" on page 1993.
7. Use the Service editor to implement a subflow for each available error handler. See "Implementing an integration service error handler subflow" on page 1994.
8. Generate your own HTTP binding for the service.

   See "Generating an integration service SOAP/HTTP binding" on page 1995. You can instead use the default HTTP binding that is generated when the service is created.
9. Optional: Generate a JavaScript client API so that you can call the integration service from a program that is running in a JavaScript environment. For more information about the JavaScript client API, see "Integration service JavaScript client API " on page 1995.

What to do next

- Deploy and test your integration service. For more information, see How do I deploy and test ?.
- View the deployed service by using the web user interface or the WebSphere Application Server administration console. For more information, see one of the following topics.
  - “Checking the results of deployment” on page 2485
  - web user interface

Integration service editor

The integration service editor is the default editor for viewing a graphical representation of a service and its operations, and for editing properties and implementing service operations. The integration service editor comprises a service overview and an interface editor. You can also define and implement operations.

Service overview

To start the integration service editor in the Editor view, open an integration service description in the Application Development view.

The **Service** tab provides a graphical overview of your integration service, showing:

- The service interface and the operations.
- The service endpoint and binding.
- The service error handlers.

The service endpoint and binding are shown on the left.

The service interface and operations are shown on the right.

Operations without an implementation are shown in the **Service** tab with a grayed-out icon.

By default, a SOAP over HTTP binding is generated when the integration service is created.

The following error handlers are created by default for each integration service:

- A Catch error handler.
- A Failure error handler.
- A Timeout error handler.

Service interface

The **Interface** tab shows the details of the operations that are defined for an integration service. Each of the operation inputs, outputs, and faults has a name and a type that is editable.

In the **Interface** tab, you can do any of the following actions:

- Define further operations. Add a new request-response or one-way operation in the Interface tab by using the controls in the Operations section. Alternatively, right-click anywhere in the editor pane.
- Remove existing operations.
- Specify names to existing operations.
- Add or remove inputs, outputs, and faults to an operation.
- Create a complex type for an input, output, or fault.

By default, a request-response operation is created when the integration service is created.

Operation implementation

You implement an operation in the integration service editor by clicking the operation that is shown in the **Service** tab. The embedded subflow editor opens for you to implement the operation as a subflow.

Use the breadcrumb navigation at the top of the editor to return to the integration service overview.

Creating an integration service from scratch

To create an integration service from scratch, define a service interface and implement its operations.

Before you begin

- Read the concept information about "Integration services" on page 1944.
- If you want to create an integration service definition that will be used by multiple integration services, create a shared library. You can then select this shared library when you complete the **New Integration Service** wizard.

If you have an existing interface that is specified in a WSDL file or designed in a Business Process Manager `.twx` file, complete the steps in one of the following topics:

- "Creating an integration service based on a WSDL file" on page 1987
- "Creating an integration service from a Business Process Manager integration service" on page 1988

About this task

To create an integration service by defining a new interface, complete the following steps:

Procedure

1. Open the **New Integration Service** wizard by using one of the following methods:
   - Click **File** > **New** > **Integration Service**.
   - In the Application Development view, click **New** > **Start by creating an integration service**.
   - Right-click the white space of the Application Development view then click **New** > **Integration Service**.
2. Enter a name for the integration service, and optionally a description.
3. To specify how you want to define the interface for your integration service, select **Define it myself using the integration service editor**.
4. Specify where you want to store your interface definition.
   - If the definition will be used by this integration service only, store the definition in the integration service project.
   - If the definition might be used by multiple integration services, store the definition in a shared library.
5. Click **Next**.
6. Optional: If the integration service requires resources that are stored in libraries, select those libraries. This step creates a reference to the libraries from the integration service.
7. Click **Finish**.

Results

An integration service is created and is shown in the Service editor. You can also expand the service in the Application Development view to view the integration service and its associated resources.

What to do next

1. Define the operations of the service interface by using the **Interface** tab of the service editor.

   Use the controls to add a request-response operation or a one-way (request) operation. You can define the inputs and outputs of each operation, and specify the name and type of each one. For more information, see "Defining the operations in a service interface" on page 1989.

2. Implement the operations that are defined in your service interface. For more information, see "Implementing an integration service operation" on page 1993.
3. Deploy and test an integration service. If you have stored your definition in a shared library, or referenced other resources in shared libraries, you must deploy the shared libraries with or before the integration service. If you update the resources in a shared library and redeploy it, those changes are available to all integration services that reference the shared library. For more information, see "Deployment rules and guidelines" on page 2465.

Creating an integration service based on a WSDL file

To create an integration service that is based on a WSDL file, define a service interface and implement its operations.

Before you begin

- Read the concept information about "Integration services" on page 1944.
- If multiple integration services might use the same WSDL file, store the WSDL file in a shared library. You can then select this shared library when you complete the **New Integration Service** wizard.

If you have an existing interface that is designed in a Business Process Manager `.twx` file, or you want to define your own interface, complete the steps in one of the following topics:

- "Creating an integration service from a Business Process Manager integration service" on page 1988
- "Creating an integration service from scratch" on page 1986

About this task

To create an integration service that is based on an existing WSDL file, complete the following steps:

Procedure

1. Open the **New Integration Service** wizard by using one of the following methods:
   - Click **File** > **New** > **Integration Service**.
   - In the Application Development view, click **New** > **Start by creating an integration service**.
   - Right-click the white space of the Application Development view, then click **New** > **Integration Service**.
2. Enter a name for the integration service, and optionally a description.
3. To specify how you want to define the interface for your integration service, select **Implement an interface already specified in an existing WSDL file**.
4. Specify how you want to use the WSDL file to implement the interface.
   - If the WSDL file will be used by this integration service only, import the file into the integration service project.
   - If you have a WSDL file that might be used by multiple integration services, use a WSDL file from a shared library.
5. Click **Next**.
6. Select the location of the WSDL file.
   - If you are importing a WSDL file, select the file from your workspace, or from an external location.
   - If you are using a WSDL file in a shared library, select the WSDL file from the relevant library in your workspace.
7. Click **Next**.
8. Select the WSDL bindings to import, and click **Next**.

   You can import only a SOAP over HTTP binding.

9. Optional: If your integration service requires resources that are stored in libraries, select the appropriate libraries, then click **Finish**.

   If, in Step 6, you decided to use a WSDL in a shared library, the reference to that library is created at that point. You do not need to refer to that library in this step.

Results

An integration service is created with the operations that are defined in the imported WSDL file. The integration service is shown in the Service editor. You can also expand the integration service in the Application Development view to view operations and associated resources.

What to do next

1. Implement the operations that are defined in your service interface. For more information, see "Implementing an integration service operation" on page 1993.
2. Deploy and test an integration service. If you have stored your WSDL file in a shared library, or referenced other resources in shared libraries, you must deploy the shared libraries with or before the integration service. If you update the resources in a shared library and redeploy it, those changes are available to all integration services that reference the shared library. For more information, see "Deployment rules and guidelines" on page 2465.

Creating an integration service from a Business Process Manager integration service

Use the **Create a new service interface from an IBM Business Process Manager integration service** option to create an integration service from a Business Process Manager `.twx` file.

Before you begin

Read the conceptual information about "Integration services" on page 1944.

If you have an existing interface that is specified in a WSDL file, or you want to define your own interface, complete the steps in one of the following topics:

- "Creating an integration service based on a WSDL file" on page 1987
- "Creating an integration service from scratch" on page 1986

Procedure

To create an integration service that is based on a Business Process Manager service, complete the following steps:

1. Open the **New Integration Service** wizard by using one of the following methods: ,
   - Click **File** > **New** > **Integration Service**.
   - In the Application Development view, click **New** > **Start by creating an integration service**.
   - Right-click the white space of the Application Development view, then click **New** > **Integration Service**.
2. Enter a name for the integration service, and optionally a description.
3. Select **Implement an interface already defined in Business Process Manager (.twx file)**, then click **Next**.
4. To select an Business Process Manager export (`.twx`) file, click **Browse** and locate the `.twx` file.
5. Select a Business Process Manager integration service, then click **Finish**.

Results

An integration service is created and is shown in the Service editor. You can also expand the service in the Application Development view to view operations and associated resources.

What to do next

1. Edit the operations that are defined in the service interface by using the **Interface** tab of the service editor.
2. Implement the operations that are defined in your service interface. For more information, see "Implementing an integration service operation" on page 1993.
3. Deploy and test an integration service. For more information, see "Deployment rules and guidelines" on page 2465.

Defining the operations in a service interface

Specify your service interface by defining the service operations, the operation types, and the operations inputs, outputs, and faults.

Before you begin

Create an integration service.

About this task

When you create an integration service in IBM App Connect Enterprise, the service opens in the service editor.

In the **Interface** tab:

- You can define further operations.
- You can remove existing operations.
- You can specify names to existing operations.
- You can add or remove inputs, outputs, and faults to an operation.
- You can create a complex type for an input, output, or fault.

For more information, see "Integration service editor" on page 1985.

Procedure

Complete the following steps to define an integration service interface:

1. Open your integration service in the Service editor by double-clicking the **Service Description** in the Application Development view, or by right-clicking and selecting **Open**.
2. Select the **Interface** tab to see the interface editor.
3. By default, a request-response operation is created when the integration service is created. Click the operation name to edit it.
4. Add or remove request-response operations and one-way operations by using the icons.
5. Add or remove inputs, outputs, and faults to operations by using the icons.

   In the **Properties** tab, click the name and type of an input, output, or fault to edit it. You can create a new complex type for an input, output, or fault by clicking the type and selecting **New...**.

Results

Your service interface is defined.

What to do next

Implement the operations that are defined in your integration service. For more information, see "Implementing an integration service operation" on page 1993.

Developing a service interface

An example of developing a service interface using the service editor.

Before you begin

Read the concept information about “Integration services” on page 1944, and create a service from scratch. See “Creating an integration service from scratch” on page 1986.

About this task

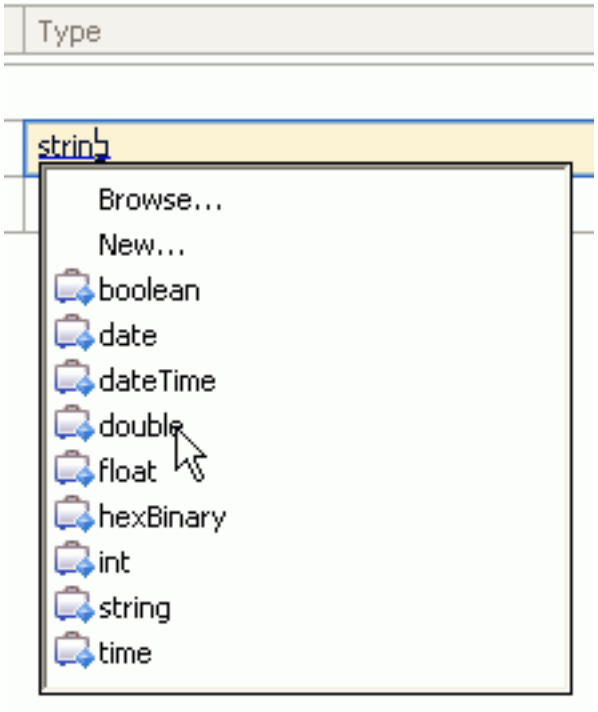
A service in IBM App Connect Enterprise is a specialized application that contains a web services solution. It defines an interface with operations that are implemented as separate subflows. The following example shows how to create an interface and define operations. The complete interface is explained in “Example of a service interface” on page 1992.

Procedure

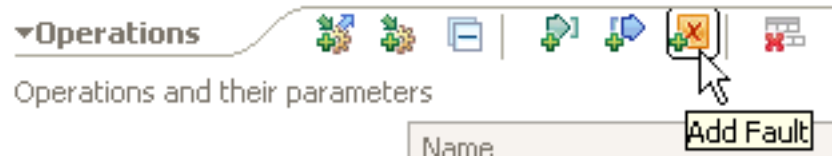
1. Open your integration service in the Service editor by double-clicking the **Service Description** in the Application Development view, or by right-clicking and selecting **Open**.
2. Display the service interface by switching to the **Interface** tab in the Service editor.
3. A default request-response operation, `operation1`, is created when the service is created. Rename `operation1` to `getApproval`.
4. Rename the inputs and outputs, and edit the types.
   a) Rename *input1* to *balance*. In the **Type** field, select the default, `string`. From the context menu, select `double`.

      The type is changed.

   b) Change *output1* to *approval* and the leave the type as `string`.



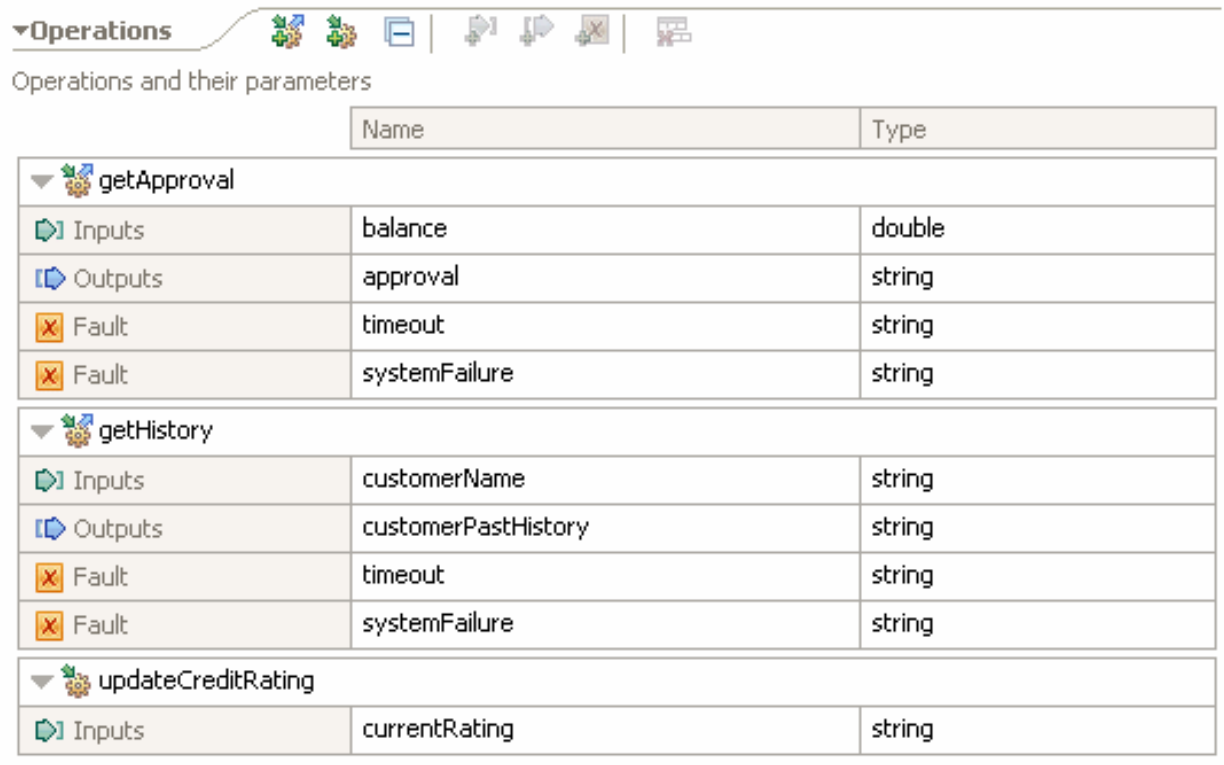
5. Add two faults to `getApproval` using the **Add Fault** icon: a *timeout* fault with a `string` type and *systemFailure*, also with a `string` type.



These faults will appear in the properties of bindings which use your interface. You can then implement the faults with fault selectors.

6. Add another request-response operation by clicking the **Add Request Response Operation** icon, or right-clicking and selecting from the context menu. A request-response operation with an input and an output is created. called `getHistory` with an input of *customerName* with a `string` type and an output of *customerHistory* with a `string` type. Select the `string` type of *customerHistory* and change the Name field to *customerPastHistory*.

7. Add the same *timeout* and *systemFailure* faults in this operation as the previous one. These faults would return error messages for timeout conditions or a system failure.
8. Add a one-way operation called `updateCreditRating`, either from the icons by selecting **Add One Way Operation** or right-clicking the canvas area in the interface editor and selecting from the context menu. Rename the input to *currentRating* with a `string` type. One-way operations send only an input as there is no response required.



	Name	Type
getApproval		
Inputs	balance	double
Outputs	approval	string
Fault	timeout	string
Fault	systemFailure	string
getHistory		
Inputs	customerName	string
Outputs	customerPastHistory	string
Fault	timeout	string
Fault	systemFailure	string
updateCreditRating		
Inputs	currentRating	string

What to do next

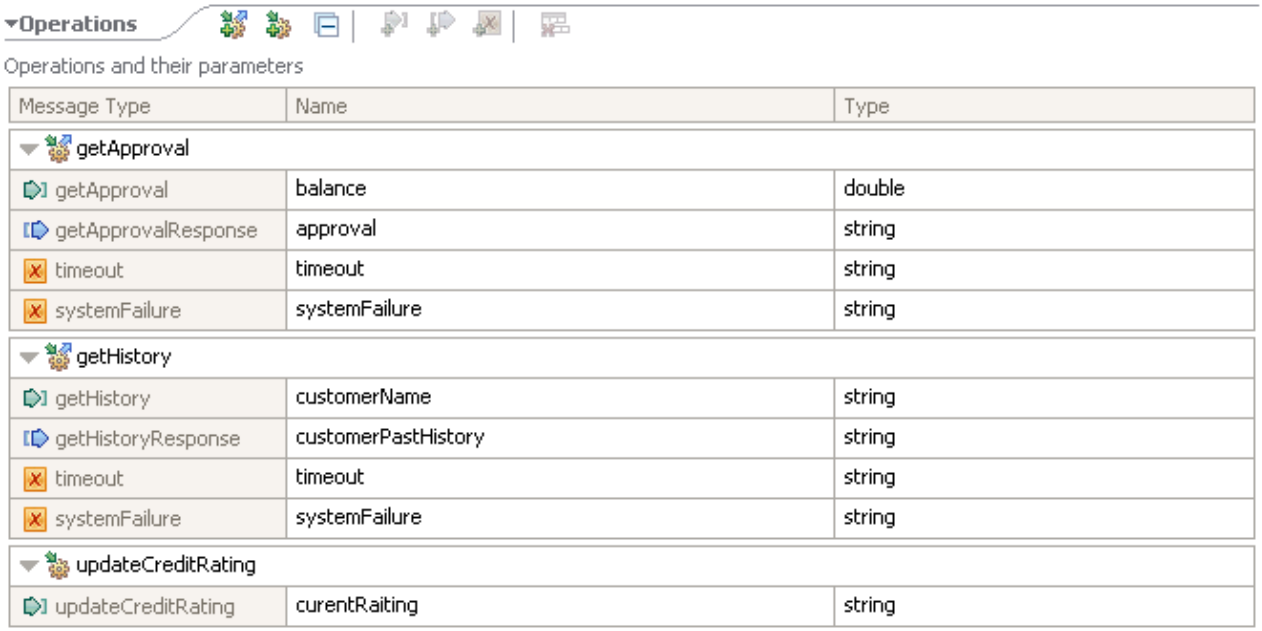
Implement the operations defined in the integration service as separate subflows. For more information, see “Implementing an integration service operation” on page 1993.

Example of a service interface

An integration service in IBM App Connect Enterprise is a specialized application with a well-defined interface, and implementation flows for each service operation. This example shows a service interface.

Read the concept information about “Integration services” on page 1944.

In the screen capture that follows, an interface has been created for a credit report. This is the interface to a component that will send a balance of a customer's account and get the approval for a transaction, get the history of the customer's account, and then update the customer's credit rating. `getApproval` and `getHistory` are request-response operations. `updateCreditRating` is a one-way operation.



Message Type	Name	Type
getApproval		
getApproval	balance	double
getApprovalResponse	approval	string
timeout	timeout	string
systemFailure	systemFailure	string
getHistory		
getHistory	customerName	string
getHistoryResponse	customerPastHistory	string
timeout	timeout	string
systemFailure	systemFailure	string
updateCreditRating		
updateCreditRating	curentRaiting	string

The following parts of the credit report interface are shown in the interface editor:

Request-response operation

`getApproval`, when invoked, sends the balance of customer account and gets the approval for a transaction. `getApproval` contains the following inputs, outputs, and faults:

- Input: `getApproval` sends as input the variable balance. The variable balance must have a double data type.
- Output: `getApproval` returns as output a variable named *approval*. The variable *approval* contains a string recommending approval (if funds are sufficient to justify granting credit) or rejection (if funds are not sufficient to extend credit to the applicant).
- Faults: `getApproval` may return one of two faults, both of which are strings describing an error condition: `timeout` is returned if the service waits for an excessive amount of time to determine approval; `systemFailure` is returned if there is communication or power failure.

Request-response operation

`getHistory`, when invoked, sends the name of a customer account and gets the history of the customer's transactions. `getHistory` contains the following inputs, outputs, and faults:

- Input: `getHistory` sends as input the variable *customerName*. The variable *customerName* must have a string data type.
- Output: `getHistory` returns as output a variable named *customerPastHistory*. The variable *customerPastHistory* contains a string with a record of past transactions.

- Fault: `getHistory` may return one of two faults, both of which are strings describing an error condition: `timeout` is returned if the service waits for an excessive amount of time to determine approval; `systemFailure` is returned if there is communication or power failure.

One-way operation

`updateCreditRating` when invoked sends the current credit rating of the customer. `updateCreditRating` has the following inputs:

- Input: `updateCreditRating` sends as input the variable *currentRating*. The variable *currentRating* must have a string data type.

By default, all added inputs and outputs of any simple type are mandatory fields. Selecting an input, output, or fault from the table opens further details about them in the properties view of the interface editor.

For an example of how to develop this interface from scratch, see "Developing a service interface" on page 1990.

Implementing an integration service operation

Implement the operations that are defined in your service interface.

About this task

You can use different editors to implement an integration service operation:

- You can implement a service operation by using the Service editor. For more information, see "Integration service editor" on page 1985.
- You can implement an operation by using the Message flow editor. For more information, see Message Flow editor.

Procedure

Complete the following steps to implement an operation:

1. Open the integration service.
   - In the Service editor, double-click the Service Description of the integration service in the Application Development view. The service **Overview** tab opens in the service editor.
   - In the Message flow editor, expand the integration service resources in the Application Development view.
2. Open the operation in an editor.
   - If you are using the Service editor, click an operation to open the embedded subflow editor.
   - If you are using the Message flow editor, double-click the subflow that represents the operation.
3. Implement the operation as a subflow. For more information, see "Developing message flows" on page 482.
   - A one-way operation has 1 input node by default.
   - A request-response operation has 1 input node and 1 output node by default.
   - Operation faults are displayed in the subflow as output nodes.

   Note: For the integration service to be called by using any binding, the response message that is built in the subflow must consist of the service payload in the XMLNSC domain. For example, if you are building a response in a Compute node, then you should write:

   ```
   SET OutputRoot.XMLNSC.NS1:echoResponse.data = 'response';
   ```

   instead of:

   ```
   SET OutputRoot.SOAP.Body.NS1:echoResponse.data = 'response';
   ```

4. Use the breadcrumb navigation at the top of the editor to return to the service overview.

What to do next

Implement the integration service error handlers. For more information, see "Implementing an integration service error handler subflow" on page 1994.

Implementing an integration service error handler subflow

Implement the error handlers that are defined in your service interface.

About this task

You edit a service interface by using the **Interface** tab of the service editor.

In the **Interface** tab:

- You can define further operations.
- You can specify names to existing operations.
- You can add inputs, outputs, and faults to an operation.
- You can implement an operation.

For more information, see "Integration service editor" on page 1985.

An integration service might encounter errors such as message failure or timeout.

IBM App Connect Enterprise provides the following error handlers to handle any errors that occur within your integration service:

- Catch error handler.
- Failure error handler.
- Timeout error handler.

You implement these error handlers as subflows:

- You implement the Catch error handler by developing the InputCatchHandler.subflow.
- You implement the Failure error handler by developing the InputFailureHandler.subflow.
- You implement the Timeout error handler by developing the InputHTTPTimeoutHandler.subflow.

Procedure

Complete the following tasks to implement an error handler subflow:

1. Open the integration service in the service editor by double-clicking the Service Description in the Application Development view.

   The service overview opens in the service editor.
2. Error handlers that are available to implement are displayed below the operations that are defined in your interface. Click on an error handler to open the embedded subflow editor and implement the error handler as a subflow.
3. Use the breadcrumb navigation at the top of the editor to return to the service overview.

What to do next

Generate a new integration service binding. For more information, see "Generating an integration service SOAP/HTTP binding" on page 1995.

Generating an integration service SOAP/HTTP binding

Learn the steps to generate a binding for your integration service.

About this task

When you create an integration service in IBM App Connect Enterprise, the service opens in the service editor. The service endpoint and binding are shown on the left of the integration service overview. The *Service Input* node represents the service endpoint. A default SOAP over HTTP binding is generated, shown next to the *Service Input* node icon.

You can also create your own binding. First, you delete the default binding, and then you can generate a new binding.

The integration service binding can be created and configured before or after you define and implement your operations.

Procedure

Complete the following steps to generate a binding:

1. In the IBM App Connect Enterprise Toolkit, open your integration service in the Service editor by double-clicking **Integration Service Description** in the Application Development view. Select the **Service** tab to see the service overview.
2. Delete the default SOAP/HTTP binding.

   Right-click the binding name and select **Remove Binding**.

   The existing binding is deleted. The service SOAPInput node icon changes to an Input node.
3. Right-click the Input node. Select **Generate** > **SOAP/HTTP Binding**. The **Generate SOAP/HTTP Binding wizard** launches.
4. Select a SOAP/HTTP binding type of either SOAP 1.1 or SOAP 1.2.
5. Enter the URL for the binding. Click **Finish**.

Results

A SOAP over HTTP binding is generated. The integration service input node icon is updated to a SOAPInput node icon, and the name of the binding is displayed next to it.

The properties of the binding are shown in the Properties view when the binding is selected in the Service editor. You can view and edit the binding properties in the Properties view.

Consumers can query your deployed integration service using a `?wsdl` query string to return the service interface. To disable this behavior, select the SOAP/HTTP binding in the Services editor, and clear `Enable support for ?wsdl` in the HTTP transport tab of the SOAP/HTTP service binding properties.

What to do next

Deploy the integration service to an integration server. For more information, see Chapter 7, “Deploying integration solutions,” on page 2463.

Integration service JavaScript client API

An integration service developer can generate a JavaScript client API from an existing integration service. The JavaScript client API provides operation functions that a JavaScript developer can call from an application that is running in a JavaScript environment.

When the integration service developer generates the JavaScript client API for an integration service, the following events occur:

- JavaScript client API code is generated from the definition of the existing integration service interface. You can use this code in JavaScript applications to call the integration service operations without

configuring the underlying communication mechanism. A JavaScript method is created for each integration service operation.

- A JSON/HTTP binding is added to the integration service so that JSON (JavaScript Object Notation) messages that are sent from the JavaScript client API can be processed by the integration service. The JSON/HTTP binding is intended for use only by the JavaScript client API.
- A web page that describes the JavaScript client API is generated. From the web page, the JavaScript developer can download or reference the JavaScript client API code and copy sample code directly into JavaScript client applications. The web page is accessed from the integration service URL.

Elements that are defined in the existing integration service, such as shared variables and Java statics, also apply to the integration service when the integration service is called by the JavaScript client API.

If the integration service is updated after the JavaScript client API is generated, then the JavaScript client API file and the associated web page are automatically updated when the integration service is saved.

Restrictions

The following restrictions apply to the JavaScript client API that is generated from an integration service description:

- Calling integration services by using the JavaScript client API is supported only from clients that are initiated either by Node.js, or by a Google Chrome web browser.
- JavaScript client API code can be generated only from an existing integration service.
- If you are deploying a web browser-based JavaScript application, then the IBM App Connect Enterprise HTTP proxy servlet must be deployed on the same web server as your JavaScript application. For information about the HTTP proxy servlet, see "HTTP proxy servlet overview" on page 141.
- The SOAP/HTTP binding is not required in order for the integration service to be called by the JavaScript client API. However, if the SOAP/HTTP binding is removed from an integration service, then the root URL for the integration service is not available and therefore the JavaScript client API code and associated web page are not accessible.

Configuring the environment for a web browser-based application that uses the JavaScript client API

To ensure that a web browser-based JavaScript application can receive messages from the integration service, you must deploy the IBM App Connect Enterprise HTTP proxy servlet onto the web application server that hosts the JavaScript application.

About this task

By default, direct communication between a web browser-based JavaScript application and an integration service is blocked by the web browser to prevent a potential cross-site scripting attack. To enable communication between the JavaScript application and an integration service, you must complete the following tasks to install and configure an HTTP proxy servlet:

Procedure

1. Configure the integration server that hosts the integration service to use the integration node listener for SOAP nodes.
2. Start an MQ listener (for example, the SYSTEM.DEFAULT.LISTENER.TCP listener) in the IBM MQ queue manager.

   For more information, search for *START LISTENER* in the IBM MQ product documentation.

3. Configure the HTTP proxy servlet with the following initialization parameters:

Parameter name	Value	Description
useClientMode	true	You should set this value to `true` even if the web application server is on the same machine as IBM App Connect Enterprise.
useQueueManagerDataInstead	*qmgr*OfConfigFile	*qmgr* is the name of your IBM MQ queue manager.
clientModeHostname	*hostname*	*hostname* is the host name or IP address for the queue manager. You can set this value to `localhost` if IBM MQ is installed on the same machine as IBM App Connect Enterprise.
clientModeChannelName	*channelname*	*channelname* is the name of the IBM MQ SVRCONN channel. The default value is SYSTEM.DEF.SVRCONN.
clientModePortNumber	*portnumber*	*portnumber* is the port number of the IBM MQ listener that you started in step 2. The default value for the SYSTEM.DEFAULT.LISTENER.TCP is 1414.

4. Deploy and configure the HTTP proxy servlet on the web application server that will host the JavaScript application.

Results

You have deployed the IBM App Connect Enterprise HTTP proxy servlet onto a web application server. A web browser-based JavaScript application (that is deployed on the same web application server as the HTTP proxy servlet) can now communicate with an integration service that is deployed on an IBM App Connect Enterprise integration server.

What to do next

"Generating a JavaScript client API for an integration service" on page 1997

Generating a JavaScript client API for an integration service

An IBM App Connect Enterprise integration service developer or administrator can generate a JavaScript client API for an existing integration service so that the integration service can be called from an application that is running in a JavaScript environment.

Before you begin

- You must have an existing integration service that uses the default SOAP/HTTP binding.
- Ensure that you are not referring to information from SOAP headers or SOAP attachments in your integration service, because this information is not available when the integration service is called by the JavaScript client API.
- If the JavaScript client API is being generated for use by a web browser-based JavaScript application, then you must configure the IBM App Connect Enterprise environment to support this usage, see "Configuring the environment for a web browser-based application that uses the JavaScript client API" on page 1996.

Procedure

Complete the following steps to generate a JavaScript client API from an existing integration service:

1. In the IBM App Connect Enterprise Toolkit, open your integration service in the integration service editor by double-clicking **Integration Service Description** in the Application Development view.
2. Click the **Service** tab.

   The integration service description is displayed, which includes the SOAP/HTTP binding.
3. Above the SOAP/HTTP binding, right-click the integration service name and click **Generate** > **JavaScript Client API**.

   The JavaScript client API for the integration service is generated and a reference to the JavaScript client API is displayed below the SOAP/HTTP binding.

   The properties of the JavaScript client API are shown in the **Properties** view when the JavaScript client API is selected in the integration service editor. You can view and edit the JavaScript client API properties from the **Properties** view.

   Note: The **Generate** > **JavaScript Client API** menu option is not available when the JavaScript client API already exists. In normal conditions, you should not need to regenerate the JavaScript client API. When you add new operations to the integration service, or update or delete existing operations, the API is automatically updated when you save the integration service. However, if you want to manually regenerate the JavaScript client API, you must remove the existing API first by right-clicking the **JavaScript Client API** entry and clicking **Remove**.

Results

- JavaScript client API files are generated from the definition of the existing integration service interface. The JavaScript client API files contain code that includes a JavaScript method for each integration service operation.
- A JSON/HTTP binding is added to the integration service so that JSON (JavaScript Object Notation) messages that are sent from the JavaScript client API can be processed by the integration service.
- A web page that describes the JavaScript client API is generated. From the web page, the JavaScript developer can download or reference the JavaScript client API code and copy sample code directly into JavaScript applications. The web page is accessed from the integration service URL.

What to do next

1. Deploy the integration service to an integration server. For more information, see Chapter 7, “Deploying integration solutions,” on page 2463.
2. Provide JavaScript developers with the integration service URL so that they can download or reference the JavaScript client API files and copy the sample code. The JavaScript developers can use the JavaScript client API to call the integration service. For more information, see “Calling an integration service by using a JavaScript client API” on page 1998.

Calling an integration service by using a JavaScript client API

When a JavaScript client API is generated from an integration service, a JavaScript developer can use the JavaScript client API to call the integration service from an application that is running in a JavaScript environment.

Before you begin

- A JavaScript client API must be generated from an existing integration service, see “Generating a JavaScript client API for an integration service” on page 1997.
- If you are developing a web browser-based JavaScript application, then you must configure your environment to use the HTTP proxy servlet, see “Configuring the environment for a web browser-based application that uses the JavaScript client API” on page 1996.

About this task

Complete the following steps to call an integration service from a JavaScript application:

Procedure

1. Understand the conventions that you must use, and the limitations that you must be aware of, when you write an application that uses the JavaScript client API to call an integration service; see "JSON conventions for calling an integration service by using a JavaScript client API" on page 2000.
2. Obtain the integration service URL from the IBM App Connect Enterprise integration service developer or administrator, or complete the following steps:
   a) In the IBM App Connect Enterprise Toolkit, click the deployed integration service in the **Integration Nodes** view.
   b) Click the **Properties** tab.

   The integration service URL is listed in the Info section.

   Note: The integration service URL is available only when the integration service has a SOAP/HTTP binding.
3. Access the integration service URL by using a web browser, and click the **JavaScript Client API** link.

   A web page is displayed. The web page contains the following items:

   - A link to the JavaScript client API file, which you can download (if you are developing a Node.js application) or reference (if you are developing a web browser-based JavaScript application). You can then use the JavaScript client API file to access the integration service.
   - Details of the JavaScript client API methods of the integration service operations, which include the input and output objects.
   - Details of the errors that might be returned from the integration service.
   - Sample code that you can add to your JavaScript application to call the JavaScript client API methods of the integration service operations, and to process any errors that are returned from the integration service.

   For more information about the contents of the web page, see "Integration service web page" on page 2001.
4. Optional: If you are developing a Node.js application, then complete the following steps:
   a) Use the Node.js package manager (npm) to install the Dojo package by changing to the Node.js installation directory and typing the following command on the command line:

   ```
   npm install dojo
   ```

   b) Download the *service_name*`.js` JavaScript client API file from the web page, where *service_name* is the name of the integration service.
   c) Create a Node.js application that imports the downloaded JavaScript client API file.
   d) Specify the host name and port of the integration service; see "Specifying the host name and port of the integration service" on page 2006.
5. Optional: If you are developing a web browser-based JavaScript application, then complete the following steps:
   a) Create a web browser-based JavaScript application.
   b) Reference the `dojo.js` module and the JavaScript client API file for the integration service; see "Referencing the dojo.js module and the JavaScript client API file" on page 2006.
6. Use the information on the integration service web page to help you complete the following tasks:
   - Create the input objects for the JavaScript client API methods of the integration service operations.
   - Use the sample code in your JavaScript application to access the JavaScript client API methods of the integration service operations.
   - Add code to your JavaScript application that handles the errors that might be returned from the integration service, and interrogates the errors for the information you require.

Results

You can write a JavaScript application to call an existing integration service.

JSON conventions for calling an integration service by using a JavaScript client API

It is possible for data to be lost when messages are automatically converted from JSON (JavaScript Object Notation) to XML, or from XML to JSON. However, you can minimize the potential for data loss during message transformation if you use documented conventions to declare and identify specific XML information within a JSON data structure definition.

When you use the JavaScript client API to call an integration service, the messages that are sent from the JavaScript client API must be converted from JSON to XML to be processed by the integration service. Messages from the integration service must be converted from XML to JSON to be retrieved by the JavaScript client API.

Since JSON has no equivalent concept for an XML attribute, conventions are used within JSON data structure definitions to distinguish between element values and attributes, and to specify how a JSON field name and associated value are represented as an XML attribute and associated value.

The following conventions are used in IBM App Connect Enterprise to handle attributes and element values:

Convention	Example

If an element has a value but no attributes, you can use standard JSON notation to indicate the value of the element.	`{"circle":"Red Circle"}`
	is equivalent to the XML string
	`<circle>Red Circle</circle>`

If an element has attributes, you must add `@` to the beginning of a JSON object name to indicate that the object is an attribute.	`{"circle":{"@color":"red"}}`
	is equivalent to the XML string
	`<circle color="red"/>`

If an element has both an attribute and a value, you must use the notation `#text` to indicate that an object is an element value.	`{"circle":{"@color":"red","#text":"Red Circle"}}`
	is equivalent to the XML string
	`<circle color="red">Red Circle</circle>`

If an element has a value but no attributes, you can use standard JSON notation to retrieve the element value.	`var myVar = {"circle":"Red Circle"};`
	`var circleValue = myVar.circle`
	The second line retrieves the value of the element "circle" from inside the JSON variable "myVar".

If an element has attributes, you must retrieve an attribute value by surrounding the attribute name with double quotation marks and square brackets.	`var myVar = {"circle":{"@color":"red"}};`
	`var circleColor = myVar.circle["@color"]`
	The second line retrieves the value of the attribute "color" from the element "circle" inside the JSON variable "myVar".

If an element has both an attribute and a value, you must retrieve the element value by using the notation `["#text"]`.	`var myVar = {"circle":`
	`{"@color":"red","#text":"Red Circle"}};`
	`var circleValue = myVar.circle["#text"]`
	The second line retrieves the value of the element "circle" from inside the JSON variable "myVar".

Limitations

Conversion of the following constructs between JSON and XML is not supported.

An XML element that contains mixed content

An output message that returns an XML element that contains an element value and nested elements cannot be converted to JSON. For example, the following XML message cannot be converted to JSON:

```
<shapes dimensions="2D">My Shapes
   <circle color="red">Red Circle</circle>
   <triangle color="yellow">Yellow Triangle</triangle>
</shapes>
```

However, the following XML message can be converted:

```
<shapes dimensions="2D">
   <circle color="red">Red Circle</circle>
   <triangle color="yellow">Yellow Triangle</triangle>
</shapes>
```

The JSON `_type` property

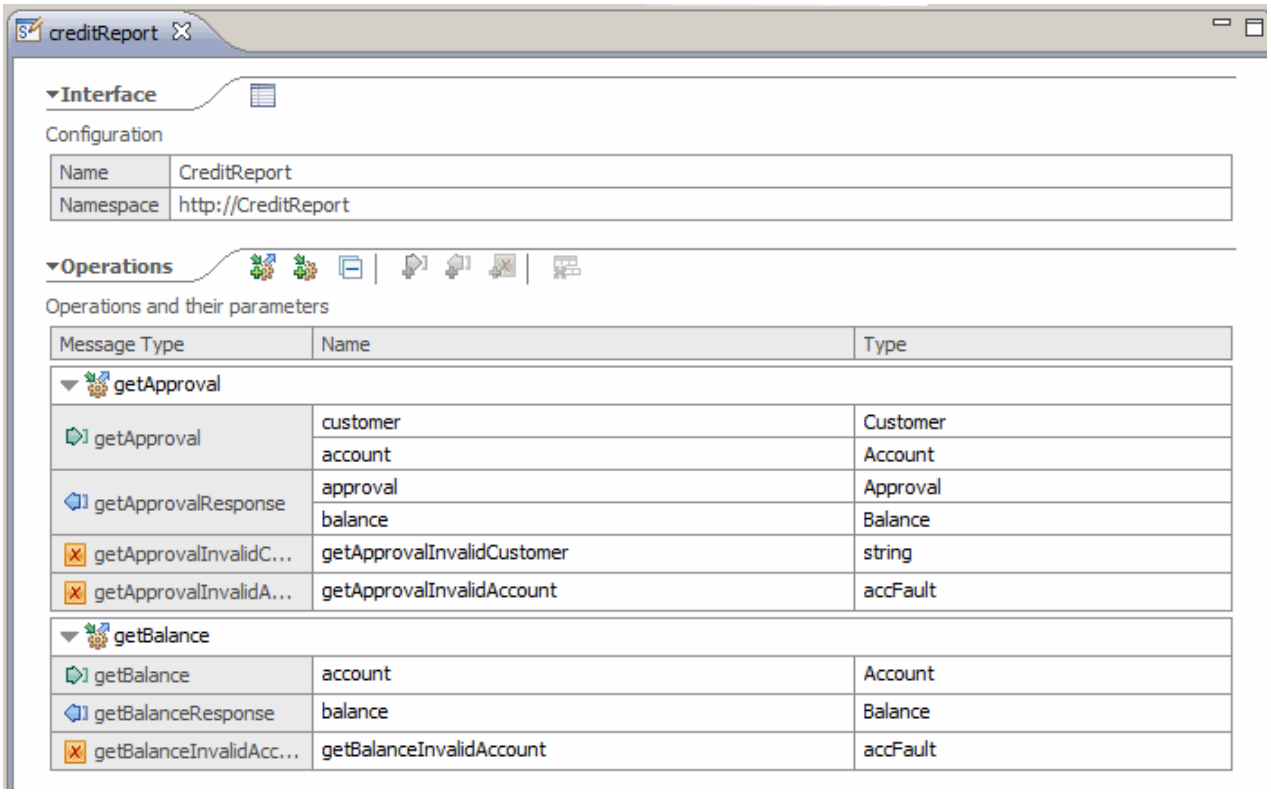
An input message that contains the JSON _type property cannot be converted to XML. For example, the following JSON message cannot be converted to XML. The integration service is still invoked, but with an XML message that does not contain anything for any JSON data that cannot be converted:

```
myCircle:{"__type":"Circle:http://www.example.org/primitiveTypesTest/",
   "x":2 ,
   "y":3 ,
   "radius":10
}
```

Integration service web page

The integration service web page contains links to the JavaScript client API file, details of the JavaScript client API methods, details of the errors that might be returned from the integration service, and sample code that a JavaScript developer can add to a JavaScript application to call the JavaScript client API.

The integration service web page is described using an example integration service. The following screen capture shows the configuration of the example integration service in the IBM App Connect Enterprise Toolkit.



When a JavaScript client API is generated, you can access the integration service URL by using a web browser, and then click the **JavaScript Client API** link to display the web page. The web page has two sections.

File

The File section contains a link to the *service_name*`.js` file, where *service_name* is the name of the integration service. In the example, the file is called `creditReport.js`. The file can be downloaded by clicking the link.

Note: You only need to download the file if you are developing a Node.js application.

Methods

The Methods section contains details of each method that is defined in the integration service. If there is more than one method that is defined in the integration service, then, at the top of the section, there are links to each method and you can click a link to navigate directly to the detail of the associated method.

For each method, there is a description, a list of the input and output objects, and any faults that are defined. The following image shows an example of a method, with the names of the input and output objects, and the names of two defined faults.

Method: IBMIntegration.creditReport.getApproval()

Description

None.

Input

customer : Customer
account : Account

Output

approval : Approval
balance : Balance

Fault

getApprovalFault1 : string

Fault

getApprovalFault2	identifier : string
	message : string

For each method, there is a coding sample that you can copy and paste into your JavaScript application. The coding sample includes the following elements:

- Sections that you might need to uncomment, depending on whether you are developing a JavaScript application for a Node.js environment, or for a web browser environment.
  - For more information about changing the host name and port that are associated with the integration service when you are developing an application for a Node.js environment, see "Specifying the host name and port of the integration service" on page 2006.
  - For more information about referencing the `dojo.js` module and the JavaScript client API file that are associated with the integration service when you are developing an application for a web browser-based environment, see "Referencing the dojo.js module and the JavaScript client API file" on page 2006.
- A comment that details the structure of the output variable, so that you know how to retrieve data from the integration service response.
- One or more comments that detail the structure of any possible faults, so that you know how to handle errors that are returned from the integration service.

  Note: There is always a fault definition (unexpected error JSON variable) to represent any exception that is generated by IBM App Connect Enterprise. For more information about the different types of exceptions you might receive, see Exception list structure and "Exception list tree" on page 516.

There might be one or more fault definitions (fault JSON variables) to represent any fault that is defined in the integration service.

- Code for the input JSON variable. You amend the values for each element to be the values that you want to send to the integration service.
- Code for the JavaScript routine that calls the integration service.

For information about the JSON conventions that are used in the JavaScript client API, and the limitations that are associated with the use of the API, see "JSON conventions for calling an integration service by using a JavaScript client API" on page 2000.

The following is the coding sample for one of the methods that is defined in the example integration service:

```
/* Uncomment these lines if you are developing in a Node.js environment.

require("http");
require("./creditReport");

IBMIntegration.creditReport.IBMContext.hostname = "myhost.company.com";
IBMIntegration.creditReport.IBMContext.port     = 7800;

*/

/* Uncomment these lines and put them in the <head> element of your HTML if you are
 developing in a browser environment.

<script type="text/javascript" src="/creditReport?resource=dojo.js"></script>
<script type="text/javascript" src="/creditReport?resource=creditReport.js"></script>

*/

/* This is an example of the output JSON variable.

var getApprovalResponseVar =
{
  "approval" :
  {
    "customer" :
    {
      "id" : "idValue" ,
      "fName" : "fNameValue" ,
      "lName" : "lNameValue"
    } ,
    "account" :
    {
      "sortCode" : "sortCodeValue" ,
      "accNo" : "accNoValue" ,
      "accType" : "accTypeValue"
    } ,
    "approved" : true
  } ,
  "balance" :
  {
    "customer" :
    {
      "id" : "idValue" ,
      "fName" : "fNameValue" ,
      "lName" : "lNameValue"
    } ,
    "account" :
    {
      "sortCode" : "sortCodeValue" ,
      "accNo" : "accNoValue" ,
      "accType" : "accTypeValue"
    } ,
    "balance" : 1.0
  }
};

*/

/*  This is an example of the unexpected error JSON variable.
    The penultimate property contains the actual exception thrown in IIB.
    Search the product documentation for 'exception list structure' to view
    the different types of exceptions and their contents.

var unexpectedErrorVar =
```

```
{
  "errName" : "Exception",
  ...
  "Label" : "{Label of the node where the exception occurred}",
  ...
  "{Exception name}" : {// penultimate property
      ... exception details ...
  },
  "Input" : {
      ... input message that caused this exception ...
  }
};


*/

/* This is an example of the fault JSON variable.

var getApprovalFault1Var =
{
  "errName" : "getApprovalFault1",
  "getApprovalFault1" : "getApprovalFault1Value"
};

*/

/* This is an example of the fault JSON variable.

var getApprovalFault2Var =
{
  "errName" : "getApprovalFault2",
  "getApprovalFault2" :   {
    "identifier" : "identifierValue" ,
    "message" : "messageValue"
  }

};

*/

/* This is an example of the input JSON variable. */

var getApprovalVar =
{
  "customer" :
  {
    "id" : "idValue" ,
    "fName" : "fNameValue" ,
    "lName" : "lNameValue"
  } ,
  "account" :
  {
    "sortCode" : "sortCodeValue" ,
    "accNo" : "accNoValue" ,
    "accType" : "accTypeValue"
  }
};


IBMIntegration.creditReport.getApproval( getApprovalVar, function( err,
 getApprovalResponseVar ){

  if (err) {
    console.log(" Failure for IBMIntegration.creditReport.getApproval() ");
    var errName = err.errName;
    if ( errName == "getApprovalFault1") {
      console.log("Fault getApprovalFault1 occurred.");
    }
    else if ( errName == "getApprovalFault2") {
      console.log("Fault getApprovalFault2 occurred.");
    }
    else if ( errName == "Exception") {
      console.log("Unexpected error occurred.");

      //To see the full details of the error, use JSON.stringify(err);

      //To retrieve only the exception, navigate to the penultimate property within the
 error
      var keys = Object.keys(err);
      console.log("Exception type: " + keys[keys.length-2]);
    }
  }
```

```
  else {
      console.log(" Success for IBMIntegration.creditReport.getApproval() ");
  }
} );
```

Specifying the host name and port of the integration service

If you are writing a JavaScript application for a Node.js client, then you must specify the host name and port that are used to call the integration service.

You must add the following lines of code to your JavaScript application, before the code that calls the integration service, so that your application can locate the integration service:

```
IBMIntegration.service_name.IBMContext.hostname = "host_name";
IBMIntegration.service_name.IBMContext.port = port;
```

where:

service_name
Specifies the name of the integration service.

host_name
Specifies the host name or IP address of the integration node where the integration service is deployed.

port
Specifies the port number of the listener that is used by the integration server where the integration service is deployed. For information about determining the listener that is used by an integration server, see “HTTP listeners” on page 795.

The lines of code are required only when the JavaScript application is run in a Node.js client. If the lines of code are included in an application that is run in a web browser, the code is ignored.

The following example shows a JavaScript application where the host name and port for an integration service named `OperationsService` are specified before the integration service is called:

```
IBMIntegration.OperationsService.IBMContext.hostname = "myhost.company.com";
IBMIntegration.OperationsService.IBMContext.port = 7800;

var multiInputVar =
{
    "input1" : "input1Value" ,
    "input2" : "input2Value"
};

IBMIntegration.OperationsService.multiInput( multiInputVar, function(errorObj, responseObj))
{
  console.log("IBMIntegration.OperationsService.multiInput()");
  console.log("Response "+JSON.stringify(responseObj));
  console.log("Response="+responseObj.output1);
});
```

Referencing the `dojo.js` module and the JavaScript client API file

If you are writing a JavaScript application for a web browser client, then you must reference the `dojo.js` module and the JavaScript client API file that are used to call the integration service.

You must add the following lines of code to your JavaScript application, before the code that calls the integration service, so that your application can locate the `dojo.js` module and the JavaScript client API file:

```
<script type="text/javascript" src="/context_root/namespace/service_name?resource=dojo.js"></
script>
<script type="text/javascript" src="/context_root/namespace/service_name?
resource=service_name.js"></script>
```

where:

context_root
Specifies the context root of the integration service (and the context root of the proxy servlet). If the context root is set to `/*`, then this entry and the preceding forward slash are omitted.

namespace
Specifies the namespace of the integration service. By default, the namespace is the name of the integration service.

service_name
Specifies the name of the integration service.

The lines of code are required only when the JavaScript application is run in a web browser client. If the lines of code are included in a Node.js application, the code is ignored.

The following example shows a JavaScript application where the `dojo.js` module and the JavaScript client API file for an integration service named `OperationsService` are specified before the integration service is called:

```
<script type="text/javascript" src="/OperationsService/OperationsService?resource=dojo.js"></
script>
<script type="text/javascript" src="/OperationsService/OperationsService?
resource=OperationsService.js"></script>

var multiInputVar =
{
    "input1" : "input1Value" ,
    "input2" : "input2Value"
};

IBMIntegration.OperationsService.multiInput( multiInputVar, function(errorObj, responseObj))
{
  console.log("IBMIntegration.OperationsService.multiInput()");
  console.log("Response "+JSON.stringify(responseObj));
  console.log("Response="+responseObj.output1);
});
```

Securing an integration service that uses a JavaScript client API

When an integration service is accessed by a JavaScript client API, you can ensure the integrity and confidentiality of the data that is sent between the integration service and the JavaScript application that calls the integration service.

Before you begin

Complete the following steps:

1. Generate a JavaScript client API for your integration service; see “Generating a JavaScript client API for an integration service” on page 1997.
2. Develop a JavaScript application that calls the integration service; see “Calling an integration service by using a JavaScript client API” on page 1998.

About this task

You can secure a SOAP/HTTP binding by using WS-Security (see “WS-Security” on page 2650), but this mechanism is not available for use by the JavaScript client API.

Note: If you developed a web browser-based JavaScript application, and IBM App Connect Enterprise and the web server that hosts the HTTP proxy servlet are on the same computer, you might want to secure only the connection between the web browser and the web server. However, if the HTTP proxy servlet receives data over SSL, then it must also forward the data to its destination over SSL. Therefore, to communicate with an integration service hosted on IBM App Connect Enterprise over a secure connection between a web browser and the HTTP proxy servlet, you must also secure the connection between the HTTP proxy servlet and the integration service.

If you want complete end-to-end security between the web browser and the integration service, you must also secure the connection to the IBM MQ queue manager. For more information about securing IBM MQ queue managers, search for *Data integrity* in the IBM MQ product documentation.

To secure the data that is sent between an integration service and a JavaScript application that calls the integration service, you must complete the following tasks:

Procedure

1. Obtain a certificate from a certificate authority, and ensure that the certificate is available in the following formats:
   - JKS format for IBM App Connect Enterprise.
   - PEM format, if you are securing a Node.js application.
   - The certificate formats that are supported by your web browser and web server, if you are securing a web browser-based JavaScript application.

   Note: You can use a self-signed server certificate for testing purposes. For more information about certificates and certificate authorities, see "Digital certificates" on page 2493.
2. Define a public key infrastructure (PKI) for IBM App Connect Enterprise and configure the PKI with the JKS keystore and truststore; see "Setting up a public key infrastructure" on page 2635.

   Note: If you have a web browser-based JavaScript application, you must define the PKI at the integration node level, or the integration node listener level (not at the integration server level or embedded listener level) because the HTTP proxy servlet is using the integration node listener.
3. Configure the integration service to use HTTPS; see "Securing integration services by using SSL" on page 2627.
4. Optional: If you are using a Node.js JavaScript application to call the integration service, then configure the Node.js application to use SSL; see "Configuring a Node.js application to access an integration service by using SSL" on page 2008.
5. Optional: If you are using a web browser-based JavaScript application to call the integration service, then configure the web browser-based application to use SSL; see "Configuring a web browser-based JavaScript application to access an integration service by using SSL" on page 2010.

Results

You have secured an integration service that uses a JavaScript client API.

Configuring a Node.js application to access an integration service by using SSL

If you are using the JavaScript client API with a Node.js application, then you can secure the data that is sent between an integration service and the JavaScript application.

Before you begin

Complete the following steps:

1. Define your public key infrastructure (PKI).
2. Configure the integration service to use SSL.

See "Securing an integration service that uses a JavaScript client API" on page 2007.

About this task

Complete the following steps to secure the data that is sent between an integration service and a Node.js application:

Procedure

1. Depending on your configuration requirements, add the following lines of code to your JavaScript application, where:

   service_name
   Specifies the name of your integration service.

   ca_cert
   Specifies the path to the CA certificate.

   client_key
   Specifies the path to the client key (the private key for your Node.js client).

   client_cert
   Specifies the path to the client certificate (the public key certificate for your Node.js client).

   client_cert_password
   Specifies the password for the client certificate.

Condition	**Add the following lines of code before you call the integration service**	**Comments**
All HTTPS configurations.	`IBMIntegration.`*`service_name`*`.IBMContext.protocol = "https";`	
The Node.js client must not check the credentials that are sent by the server.	`IBMIntegration.`*`service_name`*`.IBMContext.rejectUnauthorized = false;`	This is needed if the server provides a self-signed certificate or a CA-signed server certificate where the common name of the certificate does not match the domain name or host name of the server.

The Node.js client must validate the CA-signed certificate that is sent by the server.	`IBMIntegration.`*`service_name`*`.IBMContext.rejectUnauthorized = true;`
	
	`IBMIntegration.`*`service_name`*`.IBMContext.cacert = "`*`ca_cert`*`";`
		The Node.js client validates the CA-signed certificate by checking the CA-signed certificate against the public certificate of the CA.

The server is configured to require client authentication.	`IBMIntegration.`*`service_name`*`.IBMContext.key = "`*`client_key`*`";`
	
	`IBMIntegration.`*`service_name`*`.IBMContext.cert = "`*`client_cert`*`";`	

The client certificate uses a password.	`IBMIntegration.`*`service_name`*`.IBMContext.certpass = "`*`client_cert_password`*`";`	

For example, you might set the following properties in a Node.js client application that is using a CA-signed server certificate and a CA-signed client certificate with a password, to call an integration service named `TestService1`, where the certificates and key are stored in the Windows folder `C:\certs`.

```
IBMIntegration.TestService1.IBMContext.protocol = "https";
IBMIntegration.TestService1.IBMContext.rejectUnauthorized = true;
IBMIntegration.TestService1.IBMContext.cacert   = "C:\\certs\\ca.crt";
IBMIntegration.TestService1.IBMContext.key      = "C:\\certs\\client.key";
IBMIntegration.TestService1.IBMContext.cert     = "C:\\certs\\client.crt";
IBMIntegration.TestService1.IBMContext.certpass = "secret";
```

2. Change the port number that is associated with your integration service to the HTTPS port.

   The default HTTPS port is 7083 for the integration node listener and 7843 for the embedded listener.

   For example:

```
IBMIntegration.TestService1.IBMContext.hostname = "localhost";
IBMIntegration.TestService1.IBMContext.port = 7843;
```

Results

You have configured the Node.js client application to access the integration service by using SSL.

Configuring a web browser-based JavaScript application to access an integration service by using SSL

If you are using the JavaScript client API with a web browser-based JavaScript application, then you can secure the data that is sent between an integration service and the web browser-based JavaScript application.

Before you begin

Complete the following steps:

1. Define your public key infrastructure (PKI) at the integration node level, or the integration node listener level.
2. Configure the integration service to use SSL.

See "Securing an integration service that uses a JavaScript client API" on page 2007.

About this task

Complete the following steps to secure the data that is sent between an integration service and the browser-based JavaScript application:

Note: The steps assume that you are using the Google Chrome web browser and that the HTTP proxy servlet is deployed in an Apache Tomcat servlet container. However, the steps are similar for other web browsers and servlet containers. If you are running your servlet container behind a web server such as Apache HTTP Server, then, typically, you configure the web server to handle the SSL connections instead of configuring Apache Tomcat; see your web server documentation for information on how to configure SSL.

Procedure

1. Create a keystore for your servlet container.

   If your servlet container is on the same machine as IBM App Connect Enterprise and your servlet container uses the same certificate format, you can use the same keystore that you created for IBM App Connect Enterprise.
2. Configure the servlet container to use the keystore.

   For Apache Tomcat, complete the following steps:

   a) Edit the `server.xml` file.

      The file is found in:

      Tomcat_installation`\conf\server.xml`

      where *Tomcat_installation* is your Apache Tomcat installation directory.

   b) Uncomment the section titled `Define a SSL HTTP/1.1 Connector` and add the following lines to define the keystore and keystore password:

```
keystoreFile="keystore"
```

```
keystorePass="keystore_password"
```

where:

keystore
Specifies the path to the keystore.

keystore_password
Specifies the password of the keystore.

For example:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443
This connector uses the JSSE configuration, when using
APR, the connector should be using the OpenSSL style
configuration described in the APR documentation -->

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"

keystoreFile="c:\certs\keys.jks"
keystorePass="secret"

ReqClientAuth="false" sslProtocol="TLS" />
```

3. Save and close the `server.xml` file and restart Apache Tomcat.
4. Start the Google Chrome web browser and enter the URL for your integration service.

   Unless your keystore contains a CA certificate that the web browser can use to validate the client certificate, you get a warning message that the client certificate is untrusted.
5. To add the CA certificate, complete the following steps:
   a) Click **Settings** > **Show advanced settings** and click **Manage Certificates** in the HTTPS/SSL section.
   b) Select **Import** and use the wizard to add the PEM format of the CA certificate into the Trusted Root CA store.

Note: If you configured your integration node to support client authentication, you must also use the wizard to add the PFX format of your client certificate into the Personal store.

Results

You have configured the web browser-based JavaScript application to access the integration service by using SSL.

Developing integration solutions by using REST APIs

Use REST APIs to expose integrations as a RESTful web service that can be called by HTTP clients.

Before you begin

If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see “Developing message flows” on page 482 and “Constructing message models” on page 2133.

About this task

In IBM App Connect Enterprise, a REST API is a specialized application. A REST API contains a set of resources, and a set of operations that can be called on those resources. The operations in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser. For more information, see “REST APIs” on page 2013.

The definitions of the resources and operations that are available within a REST API are specified in Swagger or OpenAPI 3.0 documents, which are open specifications for defining REST APIs.

For more information about Swagger, see “Swagger” on page 2015.

For more information about OpenAPI 3.0, see “OpenAPI 3.0” on page 2018.

You can also learn about OpenAPI 3.0 by running the Product Tutorial "OpenAPI Specification v3 - Using an example REST API. " For more information about the product tutorials in IBM App Connect Enterprise, see Tutorials for.

IBM App Connect Enterprise also provides a set of REST nodes, which you can use to interact either synchronously or asynchronously with external REST APIs and IBM App Connect Enterprise REST APIs. For more information about using these REST nodes, see “Connecting to external REST APIs” on page 1032.

Procedure

1. Create a REST API in the IBM Integration Toolkit, by completing the steps in the relevant topic:
   - For a REST API based on Swagger, complete the steps in one of the following topics:
     - “Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit” on page 2021.
     - “Creating a REST API from an imported Swagger document” on page 2023.
   - For a REST API based on OpenAPI 3.0, complete the steps in one of the following topics:
     - “Creating a REST API from scratch with OpenAPI 3.0 specifications by using the IBM App Connect Enterprise Toolkit” on page 2022.
     - “Creating a REST API from an imported OpenAPI 3.0 document” on page 2024

     If you are importing a Swagger document, ensure that you are aware of the restrictions on the format of Swagger documents that are used to create REST APIs, as described in “Restrictions on Swagger documents” on page 2018.

     If your REST API is based on an OpenAPI 3.0 document, ensure that you are aware of the restrictions on the format of OpenAPI 3.0 documents that are used to create REST APIs. For more information, see “Restrictions on OpenAPI 3.0 documents” on page 2019.
2. If you created the REST API from scratch by using the REST API Editor in the IBM App Connect Enterprise Toolkit, you must now define the resources, models, and operations in the REST API.
   - For a REST API based on Swagger, follow the steps in “Defining resources, models, and operations in a REST API by using the REST API Editor” on page 2025.
   - For a REST API based on OpenAPI 3.0, follow the steps in “Defining resources, models, and operations in a REST API by using the OpenAPI editor” on page 2029.

   However, if you created the REST API from an imported Swagger document, you can ignore this step and go to step “3” on page 2012
3. Implement each of the operations in the REST API as a subflow, as described in the relevant topic:
   - For REST APIs based on Swagger, follow “Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents” on page 2030.
   - For REST APIs based on OpenAPI 3.0, follow “Implementing an operation for REST APIs based on OpenAPI 3.0 documents” on page 2034.
4. Optional: Implement error handling for the REST API as a set of subflows, as described in “ Implementing an error handler in a REST API” on page 2038.
5. Optional: Secure your REST API by using HTTPS for encrypting communications between client and server, as described in “Securing a REST API by using HTTPS” on page 2041.
6. Optional: Secure your REST API by authenticating users with HTTP Basic Authentication, as described in “Securing a REST API by using HTTP Basic Authentication” on page 2042.
7. Optional: If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing (CORS). For more information, see “Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing” on page 2043.
8. Optional: REST APIs are configured by default to handle JSON data. If you want to handle non-JSON data in your REST API, see “Handling non-JSON data in a REST API” on page 2039.

Results

You develop an integration solution that uses a REST API.

What to do next

- Package and deploy your REST API to an integration server, as described in "Packaging and deploying a REST API" on page 2045.
- Optional: If you want to create multiple versions of an existing REST API, see "Creating a new REST API that uses artifacts from an existing REST API" on page 2044.
- Optional: You can enable JSON Validation for your REST API by using the "Message Validation" settings in the ACE REST API Editor. For more information about JSON Validation, see "JSON validation" on page 561.
- Optional: You can push (export) your deployed REST API to an IBM API Connect server. For more information, see "Pushing REST APIs to IBM API Connect " on page 2046.

REST APIs

In IBM App Connect Enterprise, a REST API is a specialized application that can be used to expose integrations as a RESTful web service that can be called by HTTP clients. IBM App Connect Enterprise also provides a set of REST nodes, which you can use to interact either synchronously or asynchronously with external REST APIs.

A REST API describes a set of resources, and a set of operations that can be called on those resources. The operations in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser. You can also use the IBM App Connect Enterprise REST nodes to call and interact with external REST APIs. For information about using these REST nodes, see "Connecting to external REST APIs" on page 1032.

The REST API has a base path, which is similar to a context root. All resources in a REST API are defined relative to its base path. The base path can be used to provide isolation between different REST APIs, as well as isolation between different versions of the same REST API. For example, a REST API can be built to expose a customer database over HTTP. The base path for the first version of that REST API could be `/customerdb/v1`, while the base path for the second version of that REST API could be `/customerdb/v2`.

The REST API describes a set of resources. The HTTP client uses a path relative to the base path that identifies the resource in the REST API that the client is accessing. The paths to a resource can be hierarchical, and a well designed path structure can help a consumer of a REST API understand the resources available within that REST API. The following table lists some example resources for a customer database in the REST API:

Table 75. Example resources

Resource	Description
`/customers`	All the customers in the database
`/customers/12345`	Customer #12345
`/customers/12345/orders`	All orders for customer #12345
`/customers/12345/orders/67890`	Order #67890 for customer #12345

Each resource in the REST API has a set of operations that can be called by an HTTP client. An operation in a REST API has a name and an HTTP method (such as GET, POST, or DELETE). The name of the operation must be unique across all the resources in that REST API. Additionally, a single resource can have only one operation that is defined for a specific HTTP method. The combination of the path and the HTTP method that are specified by the HTTP client that is making the request is used to identify the resource and operation that is being called.

The following table lists example operations for the resource `/customers/12345`:

Table 76. Example operations		
HTTP Method	**Operation Name**	**Description**
GET	getCustomer	Retrieve the customer details from the database.
PUT	updateCustomer	Update the customer details in the database.
DELETE	deleteCustomer	Delete the customer from the database.

To call the updateCustomer operation, the HTTP client makes an HTTP PUT request to `/customerdb/v1/customers/12345`

Each operation in a REST API can also have a set of parameters that can be used by the HTTP client to pass arguments into the operation. Each parameter must be defined in the definitions for the REST API. Each parameter has a unique name and type. Several types of parameters are supported by REST APIs in IBM App Connect Enterprise:

Path parameters
Can be used to identify a particular resource. The value of the parameter is passed in to the operation by the HTTP client as a variable part of the URL, and the value of the parameter is extracted from the path for use in the operation. Path parameters are denoted by using the syntax `{paramName}` in the path to the resource. For example, the customer ID can be passed in as a path parameter named customerId:

```
/customers/{customerId}
```

Query parameters
The value of a query parameter is passed in to the operation by the HTTP client as a key value pair in the query string at the end of the URL. As an example, query parameters can be used to pass in a minimum and maximum number of results that should be returned by a particular operation:

```
/customers?min=5&max=20
```

Header parameters
The HTTP client can pass header parameters to an operation by adding them as HTTP headers in the HTTP request. As an example, header parameters might be used to pass in a unique identifier that identifies the HTTP client that is calling the operation:

```
Api-Client-Id: ff6e2c5d-42d5-4026-8f7f-d1e56da7f777
```

A single operation can define and accept multiple parameters of all three types. Additionally, depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. Operations can also send data back to the HTTP client in the response body. REST APIs in IBM App Connect Enterprise are configured by default to process JSON data, but it is possible to process data in various formats.

Each operation in the REST API must be implemented as a subflow. A subflow for an operation has a single input node, and a single output node. The implementer of the operation can build the subflow for the operation by using all of the standard message flow nodes available with IBM App Connect Enterprise.

When an HTTP client makes a call to an operation in a REST API that is deployed to an integration server, the request is automatically processed and routed to the input node in the corresponding subflow for that operation. When that subflow propagates a message to its output node, the message is automatically routed back to the HTTP client. The author of the REST API is not required to do anything to handle the HTTP transport.

To create a REST API in IBM App Connect Enterprise, you can define the models, resources, and operations from scratch, or you can use the IBM App Connect Enterprise Toolkit to import a Swagger document or an OpenAPI 3.0 document that describes the resources and operations that you want in the

REST API. For more information, see “Developing integration solutions by using REST APIs” on page 2011 and “Creating a REST API” on page 2019.

When you have created and deployed a REST API, you can push (export) the REST API definition to IBM API Connect. You can push one or more REST APIs to IBM API Connect by using either the IBM App Connect Enterprise web user interface or the **mqsipushapis** command. For more information, see “Pushing REST APIs to IBM API Connect ” on page 2046.

Swagger

Swagger is an open specification for defining REST APIs.

A Swagger document is the REST API equivalent of a WSDL document for a SOAP-based web service.

The Swagger document specifies the list of resources that are available in the REST API and the operations that can be called on those resources. The Swagger document also specifies the list of parameters to an operation, including the name and type of the parameters, whether the parameters are required or optional, and information about acceptable values for those parameters. Additionally, the Swagger document can include JSON Schema that describes the structure of the request body that is sent to an operation in a REST API, and the JSON schema describes the structure of any response bodies that are returned from an operation.

Swagger documents must be in either JSON format with a `.json` file extension, or YAML format with a `.yaml` or `.yml` file extension.

IBM App Connect Enterprise supports version 2.0 of the Swagger specification with some restrictions as described in “Restrictions on Swagger documents” on page 2018. For more information about Swagger, see Swagger. For more information about version 2.0 of the Swagger specification, see Swagger RESTful API Documentation Specification Version 2.0.

A range of third-party tools are available for you to use with Swagger documents:

Swagger Editor
: Assists you in building a Swagger document from a web browser by providing a side-by-side view of the Swagger document and the resulting REST API definitions. After you build your Swagger document, you can download it to use with IBM App Connect Enterprise. For more information, see GitHub: Swagger Editor.

Swagger UI
: Allows you to visualize and test a REST API that is defined with Swagger from any web browser. The built-in testing functions allow you to specify the inputs to an operation that is defined in that REST API, call that operation from the web browser, and inspect the results of calling that operation. For more information, see GitHub: Swagger UI.

Swagger Codegen
: Generates a Software Development Kit (SDK) in various languages, including Java, Objective-C, PHP, and Python, from a Swagger document for a REST API. The resulting SDK can be used to embed calls to the operations in that REST API into a software program that was written in one of the supported languages, without having to handle the underlying HTTP transport. For more information, see GitHub: Swagger Code Generator.

Most of the Swagger tools accept a Swagger document as a URL that points to a Swagger document that is hosted on an HTTP server. When you deploy REST APIs to IBM App Connect Enterprise, the Swagger document is published over HTTP for you to use with the Swagger tools. The scheme, host, and port number details of the Swagger document are automatically updated to match the details of the running REST API so that Swagger can be used to call the running REST API.

IBM App Connect Enterprise places some restrictions on Swagger documents, see “Restrictions on Swagger documents” on page 2018.

To see an example of a Swagger document, see Swagger petstore.

A copy of the following Swagger document, `<install root>/server/sample/restapis/swagger.json` is included with the product installation.

```
{
    "swagger": "2.0",
    "basePath": "/customerdb/v1",
    "info": {
        "title": "Customer Database",
        "description": "This is the customer database sample Swagger document included with IBM
 App Connect Enterprise.",
        "version": "1.0.0"
    },
    "definitions": {
        "Customer": {
            "properties": {
                "id": {
                    "type": "integer"
                },
                "firstname": {
                    "type": "string"
                },
                "lastname": {
                    "type": "string"
                },
                "address": {
                    "type": "string"
                }
            },
            "required": [
                "firstname",
                "lastname",
                "address"
            ]
        }
    },
    "tags": [
        {
            "name": "customers",
            "description": "Operations on customers in the customer database"
        }
    ],
    "paths": {
        "/customers": {
            "get": {
                "operationId": "getCustomers",
                "summary": "Get all customers from the database",
                "parameters": [
                    {
                        "name": "max",
                        "in": "query",
                        "description": "Maximum number of customers to get from the database",
                        "required": false,
                        "type": "integer"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "An array of customers from the database",
                        "schema": {
                            "type": "array",
                            "items": {
                                "$ref": "#/definitions/Customer"
                            }
                        }
                    }
                },
                "tags": [
                    "customers"
                ]
            },
            "post": {
                "operationId": "addCustomer",
                "summary": "Add a customer to the database",
                "parameters": [
                    {
                        "name": "body",
                        "in": "body",
                        "description": "The customer to add to the database",
                        "required": true,
                        "schema": {
                            "$ref": "#/definitions/Customer"
                        }
                    }
```

```
                ],
                "responses": {
                    "200": {
                        "description": "If the customer was successfully added to the database"
                    }
                },
                "tags": [
                    "customers"
                ]
            }
        },
        "/customers/{customerId}": {
            "get": {
                "operationId": "getCustomer",
                "summary": "Get a specified customer from the database",
                "parameters": [
                    {
                        "name": "customerId",
                        "in": "path",
                        "description": "The ID of the customer to get from the database",
                        "required": true,
                        "type": "integer"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "The specified customer from the database",
                        "schema": {
                            "$ref": "#/definitions/Customer"
                        }
                    }
                },
                "tags": [
                    "customers"
                ]
            },
            "delete": {
                "operationId": "deleteCustomer",
                "summary": "Delete a specified customer from the database",
                "parameters": [
                    {
                        "name": "customerId",
                        "in": "path",
                        "description": "The ID of the customer to delete from the database",
                        "required": true,
                        "type": "integer"
                    },
                    {
                        "name": "Authorization-Key",
                        "in": "header",
                        "description": "Provide the authorization key that permits the customer
 to be deleted from the database",
                        "required": true,
                        "type": "string"
                    }
                ],
                "responses": {
                    "200": {
                        "description": "If the customer was successfully deleted from the
 database"
                    }
                },
                "tags": [
                    "customers"
                ]
            }
        }
    }
}
```

Restrictions on Swagger documents

IBM App Connect Enterprise has some restrictions on the Swagger documents that you can use to create a REST API that are in addition to the requirements that are specified in Swagger RESTful API Documentation Specification Version 2.0.

IBM App Connect Enterprise supports version 2.0 of the Swagger specification. For information about Swagger, see Swagger. For information about version 2.0 of the Swagger specification, see Swagger RESTful API Documentation Specification Version 2.0.

IBM App Connect Enterprise places certain restrictions on the format of Swagger documents that are used to create REST APIs:

- The Swagger document must be saved in either JSON format with a `.json` file extension, or in YAML format with a `.yaml` or `.yml` file extension.
- You must enter a value for the **operationId** field in the Operation Object, and the value of the **operationId** field must be unique across all operations that are defined in this REST API. IBM App Connect Enterprise uses the operation ID as a unique name to refer to the operation.
- The base paths of all REST APIs deployed to a single integration server must be unique. You cannot deploy a REST API to an integration server if there is already another REST API deployed with the same base path.
- Form data parameters:
  - When IBM App Connect Enterprise is implementing a REST API, form data parameters are permitted in Swagger documents, but no support is provided for parsing form data parameters.
  - When a REST API is being consumed through a RESTRequest node, form data parameters are not supported.
- The `"file"` JSON schema type is permitted in Swagger documents but is not supported by IBM App Connect Enterprise; you cannot import a Swagger document that includes a reference to the `"file"` JSON schema type.
- The JSON schema **`"$ref"`** keyword is supported with the following restrictions:
  - Only internal file references that have the form `"$ref" : "#/definitions/Pet"` are allowed.
  - Recursive self references can be imported into the IBM App Connect Enterprise Toolkit. However, if you use a Mapping node, the Graphical Data Mapping editor cannot be used to dynamically discover the depth of recursion. If recursive self references have been imported, use mapping only when you want to move the structure, by using a **Move** transform, from input to output.

When you are using a Swagger document in a message map, ensure that you satisfy the requirements described in JSON schema requirements for message maps.

OpenAPI 3.0

OpenAPI 3.0 is an open specification for defining REST APIs.

The OpenAPI 3.0 Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs, which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

An OpenAPI 3.0 definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

IBM App Connect Enterprise supports version 3.0 of the OpenAPI specification with some restrictions. For more information, see https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.0.md and “Restrictions on OpenAPI 3.0 documents” on page 2019.

Restrictions on OpenAPI 3.0 documents

IBM App Connect Enterprise has some restrictions on the OpenAPI 3.0 documents that you can use to create a REST API. These restrictions are in addition to the requirements that are specified in the OpenAPI 3.0 specification.

Consider the following limitations and restrictions when you use OpenAPI 3.0 REST APIs:

- You can import and implement OpenAPI 3.0 documents that contain external references, but you cannot edit these documents by using theIBM App Connect Enterprise Open API Editor.
- You can use the IBM App Connect Enterprise Toolkit to edit only one OpenAPI 3.0 document at a time.
- The base paths of all REST APIs deployed to a single integration server must be unique. You cannot deploy a REST API to an integration server if another REST API is deployed with the same base path.
- When the OpenAPI Editor is open, the REST API Editor is set to 'read only' mode. Only the **Create**, **Edit**,and **Delete** subflow buttons in the OpenAPI Editor are active.
- You can use OpenAPI 3.0 documents that contain multiple server objects, but the base URL in each object must be the same. You cannot use server templating, in which server variables are used to replace values in a URL template in the server object.
- Some aspects of the OpenAPI 3.0 specification are not currently supported by the user interface. In these circumstances, you can use the **source view** of the Open API editor to edit the specification text directly.
- The user interface does not currently support referencing a request body component from the request body of an operation. If you want to reference a request body component, add the reference to the request body of the operation directly in the OpenAPI source, as shown in the following example:

  ```
  requestBody: $ref: '#/components/requestBodies/request_body_component_name'
  ```

  The reference is confirmed on the details page for the request body in the user interface.
- Some OpenAPI 3.0 objects are ignored when you create REST API projects in App Connect Enterprise.
  - Cookie data parameters are allowed in OpenAPI 3.0 documents by IBM App Connect Enterprise, but no support is provided for parsing the HTTP Input header **Cookie** field into `LocalEnvironment.REST.Input.Parameters`.
  - Parameters of complex type are allowed in OpenAPI 3.0 documents, but IBM App Connect Enterprise does not currently support parsing or mapping complex type parameters.

Creating a REST API

To create a REST API in IBM App Connect Enterprise, you can define the models, resources, and operations in the REST API Editor, or the OpenAPI editor. Alternatively, you can use the IBM App Connect Enterprise Toolkit to import a Swagger document or an OpenAPI 3.0 document that describes the resources and operations that you want in the REST API.

About this task

Both methods of creating a REST API have advantages, and the most appropriate method depends on your circumstances.

Creating a REST API from scratch by using Swagger specifications

You can use the REST API editor that is integrated into the IBM App Connect Enterprise Toolkit to create a REST API from scratch. The REST API editor enables you to graphically create a REST API by defining the resources, operations, and parameters for that REST API. You can also use the REST API editor to define models that represent the structure of JSON request and response bodies for the operations in the REST API. The models that you define with the REST API editor can then be used by the Graphical Data Mapping editor to implement the operations in the REST API.

The REST API editor creates and modifies a Swagger document that is compliant with version 2.0 of the Swagger specification. The Swagger document can be used with the Graphical Data Mapping editor to

implement operations in the REST API as well as a wide range of external tools. For example, Swagger UI, Swagger Codegen, and IBM API Management. However, the REST API editor does not support the full Swagger specification, and cannot be used to set some properties, such as validation attributes, on objects in the Swagger document.

Use the REST API editor to create a REST API from scratch in the following circumstances:

- You have limited or no knowledge of the Swagger specification.
- You prefer to use the graphical tools that are integrated into the IBM Integration Toolkit, rather than using an external editor to create a Swagger document.

Creating a REST API from scratch by using OpenAPI 3.0 specifications

You can use the OpenAPI editor that is integrated into the IBM App Connect Enterprise Toolkit to create a REST API from scratch. By using the OpenAPI editor, you can graphically create a REST API by defining the resources, operations, and parameters for that REST API. You can also use the OpenAPI editor to define models that represent the structure of JSON request and response bodies for the operations in the REST API. The models that you define with the OpenAPI editor can then be used by the Graphical Data Mapping editor to implement the operations in the REST API.

The OpenAPI editor creates and modifies an OpenAPI 3.0 document. The OpenAPI 3.0 document can be used with the Graphical Data Mapping editor to implement operations in the REST API.

Use the OpenAPI editor to create a REST API from scratch by using OpenAPI 3.0 specifications in the following circumstances:

- You have limited or no knowledge of the OpenAPI 3.0 specification.
- You prefer to use the graphical tools that are integrated into the IBM Integration Toolkit, rather than using an external editor to create an OpenAPI 3.0 document.

Creating a REST API by importing a Swagger document

You can use an external editor, such as Swagger Editor, to create a Swagger document that defines the REST API that you want to implement. You can then create a REST API by importing this Swagger document by using the IBM App Connect Enterprise Toolkit. All the resources, operations, and parameters are loaded from the Swagger document and are viewable from the REST API editor in the IBM App Connect Enterprise Toolkit. The models are also loaded from the Swagger document, and those models can then be used with the Graphical Data Mapping editor to implement operations in the REST API.

The REST API editor in the IBM Integration Toolkit also allows you to modify the REST API, such as adding new resources, and it modifies the imported Swagger document automatically. However, the REST API editor does not support the full Swagger specification, and might be unable to display or modify the full contents of an imported Swagger document.

Create a REST API by importing a Swagger document in the following circumstances:

- You have good knowledge of the Swagger specification, and you create a Swagger document for the REST API that you plan to implement in IBM App Connect Enterprise.
- A separate member of your team, such as an API Designer, is creating a Swagger document for the REST API that you plan to implement in IBM App Connect Enterprise.
- You require the use of all properties in the Swagger specification; for example, the validation attributes such as maxLength, uniqueItems, and allOf, which are available when you define models.

Creating a REST API by importing an OpenAPI 3.0 document

You can use an external editor to create an OpenAPI 3.0 document that defines the REST API that you want to implement. You can then create a REST API by importing this OpenAPI 3.0 document by using the IBM App Connect Enterprise Toolkit. All the resources, operations, and parameters are loaded from the OpenAPI 3.0 document and are viewable from the OpenAPI editor in the IBM App Connect Enterprise Toolkit. The models are also loaded from the OpenAPI 3.0 document, and those models can then be used with the Graphical Data Mapping editor to implement operations in the REST API.

The OpenAPI editor in the IBM Integration Toolkit also allows you to modify the REST API, such as adding new resources, and it modifies the imported OpenAPI 3.0 document automatically.

Create a REST API by importing an OpenAPI document in the following circumstances:

- You have good knowledge of the OpenAPI 3.0 specification, and you create an OpenAPI 3.0 document for the REST API that you plan to implement in IBM App Connect Enterprise.
- A separate member of your team, such as an API Designer, is creating an OpenAPI 3.0 document for the REST API that you plan to implement in IBM App Connect Enterprise.
- You require the use of all properties in the OpenAPI 3.0 specification.

Procedure

Complete the steps in one of the following tasks to create a REST API in IBM App Connect Enterprise:

- "Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit" on page 2021
- "Creating a REST API from scratch with OpenAPI 3.0 specifications by using the IBM App Connect Enterprise Toolkit" on page 2022
- "Creating a REST API from an imported Swagger document" on page 2023.
- "Creating a REST API from an imported OpenAPI 3.0 document" on page 2024.

  If you are importing a Swagger document, ensure that you are aware of the restrictions on the format of Swagger documents that are used to create REST APIs, as described in "Restrictions on Swagger documents" on page 2018.

What to do next

- If you created the REST API from scratch by defining resources, models, and operations in the IBM App Connect Enterprise Toolkit, you must now define the operations in the REST API Editor or the OpenAPI editor before you implement them. For information, see "Defining resources, models, and operations in a REST API by using the REST API Editor" on page 2025, and "Defining resources, models, and operations in a REST API by using the OpenAPI editor" on page 2029.
- If you create the REST API from an imported Swagger document, you must then implement the operations, as described in "Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents" on page 2030.

Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit

You can create a REST API by defining the resources, models, and operations in the IBM App Connect Enterprise Toolkit.

About this task

You can use the REST API editor to create a REST API from scratch with Swagger specifications. Alternatively, you can Create a REST API from scratch with OpenAPI specifications by using the IBM App Connect Enterprise Toolkit.

You can also Create a REST API by importing a Swagger document or Create a REST API by importing an OpenAPI document.

Procedure

To create a REST API by defining the resources, models, and operations, complete the following steps:

1. Open the **Create a REST API** wizard by clicking **File** > **New** > **REST API** .

2. Enter a name for the REST API.

   The name that you specify is used as the name of the project in the IBM App Connect Enterprise Toolkit.
3. Select **Create a REST API and define resources and operations yourself**.
4. Select **Swagger 2.0** from the drop-down menu.

   The **API base path** field and the **Version** field are completed automatically.
5. To finish creating the REST API, click **Finish**.

   The **REST API Editor** for the new REST API opens automatically, which you must now use to define resources, models, and operations, as described in "Defining resources, models, and operations in a REST API by using the REST API Editor" on page 2025.

Results
The REST API is created.

What to do next
Define the operations in the REST API and then implement them, by completing the steps in the following topics:

1. "Defining resources, models, and operations in a REST API by using the REST API Editor" on page 2025
2. "Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents" on page 2030

Creating a REST API from scratch with OpenAPI 3.0 specifications by using the IBM App Connect Enterprise Toolkit

You can create a REST API from scratch with OpenAPI 3.0 specifications by defining the resources, models, and operations in the OpenAPI editor that is integrated with the IBM App Connect Enterprise Toolkit.

About this task
You can use the OpenAPI editor to create a REST API from scratch with OpenAPI 3.0 specifications. Alternatively, you can Create a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit.

You can also Create a REST API by importing a Swagger document, or Create a REST API by importing an OpenAPI3 document.

Procedure
To create a REST API by defining the resources, models, and operations, complete the following steps:

1. Open the **Create a REST API** wizard by clicking **File** > **New** > **REST API** .
2. Enter a name for the REST API.

   The name that you specify is used as the name of the project in the IBM App Connect Enterprise Toolkit.
3. Select **Create a REST API and define resources and operations yourself**.
4. Select **OpenAPI3** from the menu.

   The **API base path** field and the **Version** field are completed automatically.
5. To finish creating the REST API, click **Finish**.

   The REST API Editor for the new REST API opens automatically, but you can edit the generated values. You must now use the OpenAPI editor to define resources, models, and operations, as described in "Defining resources, models, and operations in a REST API by using the OpenAPI editor" on page 2029.

Results
The REST API is created.

What to do next
Define the operations in the REST API and then implement them, by completing the steps in the following topics:

1. "Defining resources, models, and operations in a REST API by using the OpenAPI editor" on page 2029
2. "Implementing an operation for REST APIs based on OpenAPI 3.0 documents" on page 2034

Creating a REST API from an imported Swagger document

To create a REST API in IBM App Connect Enterprise, you can import a Swagger document by using the IBM App Connect Enterprise Toolkit.

Before you begin
Create a Swagger document that describes the resources and operations that you want in the REST API. Ensure that the Swagger document adheres to the guidelines in the Swagger specification, and that it satisfies the requirements of IBM App Connect Enterprise. For more information, see Swagger RESTful API Documentation Specification Version 2.0 and "Restrictions on Swagger documents" on page 2018.

About this task
To create a REST API in IBM App Connect Enterprise, you can use the **Create a REST API** wizard in the IBM App Connect Enterprise Toolkit to import a Swagger document. You must then create the REST API and other project artifacts that are needed to implement and deploy the new REST API. Alternatively, you can create a REST API by defining the models, resources, and operations from scratch, as described in "Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit" on page 2021.

Procedure
To create a REST API by importing a Swagger document, complete the following steps:

1. Open the **Create a REST API** wizard by clicking **File** > **New** > **REST API** .
2. Enter a name for the REST API.

   The name that you specify is used as the name of the project in the IBM App Connect Enterprise Toolkit.
3. Select **Import resources and operations defined in a Swagger document** and then click **Next**.
4. Select the path to the Swagger document that describes the resources and operations that you want in the REST API.

   You can import the Swagger document from the file system or from an existing project in the workspace.

   The file is validated for use in a REST API. If any errors are found while the selected Swagger document is validated, those validation errors are displayed in the wizard. If validation errors are found in the selected Swagger document, you cannot proceed.
5. Optional: To review the list of resources and operations that are contained in the selected Swagger document, click **Next**.
6. To finish creating the REST API, click **Finish**.

   The **REST API Description** for the new REST API opens automatically, which you can use to implement operations as described in "Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents" on page 2030.

Results
The REST API is created.

What to do next

Implement each of the operations in the REST API as a subflow, see "Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents" on page 2030.

Creating a REST API from an imported OpenAPI 3.0 document

To create a REST API in IBM App Connect Enterprise, you can import an OpenAPI 3.0 document by using the IBM App Connect Enterprise Toolkit.

Before you begin

Create an OpenAPI 3.0 document that describes the resources and operations that you want in the REST API. Ensure that the OpenAPI 3.0 document adheres to the guidelines in the OpenAPI 3.0 specification, and that it satisfies the requirements of IBM App Connect Enterprise. For more information, see https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.0.md and "Restrictions on OpenAPI 3.0 documents" on page 2019.

About this task

To create a REST API in IBM App Connect Enterprise, you can use the **Create a REST API** wizard in the IBM App Connect Enterprise Toolkit to import an OpenAPI 3.0 document. You must then create the REST API and other project artifacts that are required to implement and deploy the new REST API. Alternatively, you can create a REST API by defining the models, resources, and operations from scratch, as described in "Creating a REST API from scratch with OpenAPI 3.0 specifications by using the IBM App Connect Enterprise Toolkit" on page 2022.

Procedure

To create a REST API by importing an OpenAPI 3.0 document, complete the following steps:

1. Open the **Create a REST API** wizard by clicking **File** > **New** > **REST API** .
2. Enter a name for the REST API.

   The name that you specify is used as the name of the project in the IBM App Connect Enterprise Toolkit.
3. Select **Import resources and operations defined in a RESTAPI document** and then click **Next**.
4. Click **Next**.
5. Select the path to the OpenAPI 3.0 document that describes the resources and operations that you want in the REST API.

   You can import the OpenAPI 3.0 document from the file system or from an existing project in the workspace.

   The file is validated for use in a REST API. If any errors are found while the selected OpenAPI 3.0 document is validated, those validation errors are displayed at the top of the wizard. You cannot proceed if validation errors are found in the selected OpenAPI 3.0 document.
6. Optional: To review the list of resources and operations that are contained in the selected OpenAPI 3.0 document, click **Next**.
7. To finish creating the REST API, click **Finish**.

   The **REST API Description** for the new REST API opens automatically, which you can use to implement operations as described in "Implementing an operation for REST APIs based on OpenAPI 3.0 documents" on page 2034.

Results

The REST API is created.

What to do next

Implement each of the operations in the REST API as a subflow, see "Implementing an operation for REST APIs based on OpenAPI 3.0 documents" on page 2034.

Defining resources, models, and operations in a REST API by using the REST API Editor

Use the **REST API Editor** to graphically define resources, models, and operations in a REST API.

Before you begin

Create a REST API in the IBM App Connect Enterprise Toolkit, as described in "Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit" on page 2021.

About this task

Use the sections in the **REST API Editor** to define your REST API.

Procedure

1. In the IBM App Connect Enterprise Toolkit, navigate to your REST API project.
2. Open the REST API Description by double-clicking the REST API project or by right-clicking and selecting **Open**.

   The REST API editor for Swagger documents opens and provides a view of the API details, it Resources and Operations, Error handling, Security, and Message Validation Settings.

To define resources, models, and operations in a REST API, complete the following steps:

3. Use the **Header** section of the REST API Editor to display and modify general information about the REST API.

   The **REST API base URL** field displays the current base path for the REST API. All resources in a REST API are defined relative to its base path. You can edit this field to modify the base path of the REST API. The value of this field must be a valid URL path segment, and must not contain any variable elements.

   The `Title` field displays the current title of the REST API. The Title is distinct from the REST API project name, and is defined as a property in the Swagger document. The Title is used when you push a REST API into IBM API Management.

   The `Version` field displays the current version of the REST API. It is defined as a property in the Swagger document. The Version is used when you push a REST API into IBM API Management.

   The header section also includes an example REST API base URL. The *hostname* and *port number* portions of this URL must be replaced with the hostname and port number of the integration server that the REST API is deployed to. To start an operation that is defined in this REST API by using an HTTP client, you must combine the REST API base URL, the path of the operations resource, and the HTTP method for the operation that you are starting. Depending on the operation that is being started, you might also need to pass extra parameters, or a request body.
4. Use the **Resources** section to manipulate the resources for your REST API:

   Use this button to create a new resource in your REST API. When you click this button, a dialog is displayed in which you can specify the resource path and select the operations that you

want to add to the resource: GET, POST, PUT, DELETE, OPTIONS, HEAD, and PATCH. When you click **OK**, the new resource and the operations for that resource are added to the REST API.

You can also add and remove operations from a resource after it is created. Use the  button on the right side of the resource to add more operations to an existing resource. Use the  button on the right side of the operation to remove that operation from its resource.

 Use this button to modify your resource path. When you click this button, a dialog is displayed in which you can modify the resource path.

 Use this button to expand all operations under each resource.

 Use this button to collapse all operations under each resource.

5. When you add an operation under a resource, you can manipulate it.

   An operation can have several different kinds of parameters, and 0 or more responses, which are shown in the following sections:

   - **Path, Query, and Header parameter**

     Use the Add button  and the Delete button  to add or delete Header and Query parameters, and then set their Name, Type, and Data Type, and optionally, the Format, Required, and Description values.

     The `Name` and `Type` fields of a Path parameter cannot be changed, and the name must match the path template in the resource. Path parameters can be added or removed only by editing the path template in the Resource.

     The Data Type of a Path, Query, or Header parameter can be set only to a primitive data type, such as string, number, integer, or Boolean. These primitive data types correspond with those in the Swagger specification, and map to the possible primitive types in a JSON document. You cannot set the Data Type of a Path, Query, or Header parameter to that of a Model defined in the REST API. The Data Type of a Path, Query, or Header parameter is not currently used by the IBM App Connect Enterprise run time, and the parameters are provided to the nodes in the subflow as character data. The Graphical Data Map models them as strings. However, it is advisable to set the parameter correctly for use with Swagger tooling.

     The `Format` field is an optional free-form description of the format of the data that is expected for values of this parameter. For example, you can use formats that are mentioned in the Swagger specification such as `int32`, `double`, or `date-time`. Alternatively, you can include your own format names, such as `customer-id`. The Format of a Path, Query, or Header parameter is not currently used by the IBM App Connect Enterprise run time; however, it is advisable to set it correctly for use with Swagger tooling.

     The `Required` field states whether the parameter must be specified by HTTP clients that start this operation. Path parameters must always be specified in the request URL by an HTTP client. This field cannot be changed for Path parameters. Query and Header parameters can be marked as optional by deselecting this checkbox. The presence of a Query or Header parameter is not currently validated

by the IBM App Connect Enterprise run time. However, it is advisable to set it correctly for use with Swagger tooling.

The Swagger Form parameter type is not supported in IBM App Connect Enterprise.

- **Request Body**

  Use the Add request button  and the Delete button  to add or delete a request body, if a request body is allowed for the HTTP method of the selected operation. For example, `GET`, `HEAD`, `DELETE`, and `OPTIONS` requests do not permit request bodies.

  A request body is usually a JSON document, and the structure of that JSON document can be defined in a Model. The structure of a request body is not currently validated by the IBM App Connect Enterprise run time. However, the Model definition can be used with a Mapping node to graphically implement the REST API operation. It is advisable to define Models for all of the possible JSON request bodies in the API, for use with Swagger tooling.

  The `Schema type` for Request includes all primitive data types, such as `string`, `number`, `integer`, or `boolean`, plus all models defined in the API.

- **Response Body**

  Use the Add response button  and the Delete button  to add or delete a response, if a response body is permitted for the HTTP method of the selected operation. For example, `HEAD` requests do not return a response body.

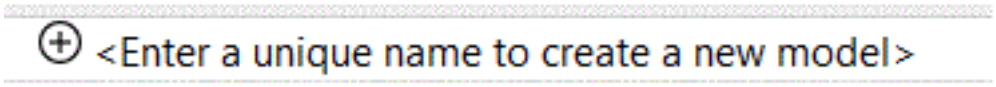
  Set the HTTP `status code number` for this response as the value of the `Response Status` field. The HTTP status code must be a valid HTTP status code as described by RFC 7231 or the IANA Status Code Registry. You can define multiple responses for a single operation, but the HTTP status code of a response must be unique within that operation.

  A response body is usually a JSON document, and the structure of that JSON document can be defined in a Model. The structure of a response body is not currently validated by the IBM App Connect Enterprise run time. However, the Model definition can be used with a Mapping node to graphically implement the REST API operation. It is advisable to define Models for all of the possible JSON request bodies in the API for use with Swagger tooling.

  The `Schema type` for Response includes all primitive data types, such as `string`, `number`, `integer`, or `boolean`, plus all models defined in the API. Additionally, you can specify if this operation returns an array of values, where the type of each item in that array is defined by the **Schema type** field. If you specify that the response is an array, and the Schema type refers to a Model that is itself an array, the Response is defined as returning an array of arrays.

6. Use the **Model Definitions** section of the editor to define the JSON data that you can use for the Request and Response `Schema type` in your operations.

   To create a new Model data type, click the Add button next to the **Model Definitions** field, and enter a name for your new JSON data type:

   You can define models of type `object`, `string`, `integer`, `number`, `boolean`, and `null`.

   Note: The file data type is not supported by IBM App Connect Enterprise; you cannot set a parameter or model type as `file`.

   To create an object Model when you have entered its name, use the **Add a child for selection** button

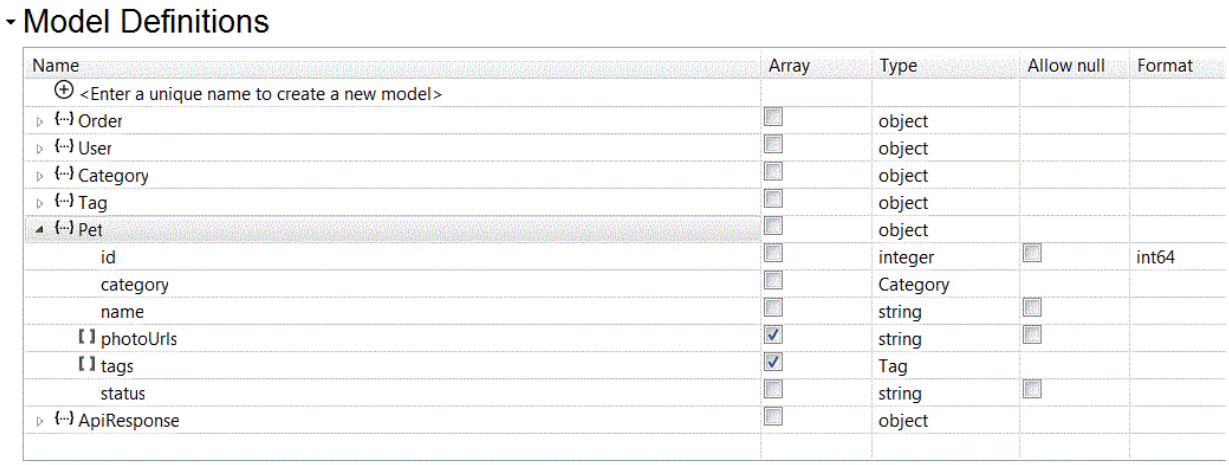
   

   , located in the **Model Definitions** section, to add the first child. You can then add more children to build up the required structure:

   You can specify if the data type must be an array, whether it is required, whether it allows null, and you can specify a format. **Format** is a free form text field, which you can use to describe the format of the data. For example, you can have a string with a format of `date`. For more information, see http://swagger.io/specification/#dataTypeFormat.

   The following example shows the definition of a *Pet* schema type:

   This *Pet* Model schema type could be used in a GET operation to produce a JSON data message with the following format:

```
{
 "id": 1001,
 "category": {
  "id": 10,
  "name": "Cat"
 },
 "name": "Moggy",
 "photoUrls": ["http:\/\/something.com\/images?q=tbn:XmoggyJjB2XhAqq97VzJP"],
 "tags": [{
  "id": 11,
  "name": "Moggy"
 }],
 "status": "pending"
```

```
}
```

You can also define a GET for returning multiple Pets by setting the operation Schema type to *Pet* and checking the `Array` option:

While the Open API editor is active in a separate window to allow editing the resources, models, and operations theIBM App Connect Enterprise Toolkit REST API editor for Swagger documents is in a read-only mode. You must close the REST API API editor to re-enable the ACE Toolkit REST API editor for Swagger documents.

Results

The resources, models, and operations are defined in the REST API.

What to do next

Implement each of the operations in the REST API as a subflow, as described in "Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents" on page 2030.

Defining resources, models, and operations in a REST API by using the OpenAPI editor

Use the **OpenAPI Editor** to graphically define resources, models, and operations in a REST API.

Before you begin

Create a REST API in the IBM App Connect Enterprise Toolkit, as described in "Creating a REST API" on page 2019.

About this task

Use the sections in the **OpenAPI Editor** to define your REST API.

The **OpenAPI Editor** has a **Navigation view** to position into the various objects of the OAI3 document and to create and delete those objects. Selecting an object in the Navigation view displays a **Form View** where you can view, create, and change the content of that object. There is also a **Source View**, which you can use to display the raw OpenAPI 3.0 document. The Source View is accessed by clicking the **Source View** icon.

Procedure

To define resources, models, and operations in a REST API, complete the following steps:

1. Click **Edit API document in OpenAPI Editor** to open the OpenAPI editor in a separate window.
2. Click the '+' sign next to **Path** to open the **Path editor**
3. Enter a path name into the **Path** field.
4. Optional: Enter a summary in the **Summary** field.
5. Optional: Enter a description in the **Description** field.
6. Click **Create** to create the OpenAPI 3.0 document.
7. Click the '+' sign next to **Operations** to add an operation.
8. Select an operation from the menu.
9. Optional: Enter a summary in the **Summary** field.
10. Optional: Enter a description in the **Description** field.
11. Click **Create** to create the operation.

12. Click **Save** to save the operation.
13. Continue to update the REST API by adding items from the list of components, such as Servers, Tags, Parameters.
14. When you finish updating the RESTAPI, click **Save**.
15. Close the OpenAPI editor.

    If you return to the IBM App Connect Enterprise Toolkit, you can see that the **Resources and Operations** section of the RESTAPI is now populated.

Results

The resources, models, and operations are defined in the REST API.

What to do next

Implement each of the operations in the REST API as a subflow, as described in "Implementing an operation for REST APIs based on OpenAPI 3.0 documents" on page 2034.

Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents

Use the **REST API Editor** in the Application Development view to implement operations in a REST API.

Before you begin

- Create a REST API based on a Swagger 2.0 specification document in the IBM App Connect Enterprise Toolkit, as described in "Creating a REST API" on page 2019.
- If you created the REST API from scratch by defining resources, models, and operations (as described in "Creating a REST API from scratch with Swagger specifications by using the IBM App Connect Enterprise Toolkit" on page 2021, you must also define the operations in the REST API Editor. For more information, see "Defining resources, models, and operations in a REST API by using the REST API Editor" on page 2025.

About this task

Operations in a REST API are implemented as a subflow, and you must use the REST API editor to create an empty subflow for each operation. You can then implement the operation by adding any of the standard IBM App Connect Enterprise message flow nodes to the subflow.

If you do not implement an operation, you can still deploy the REST API to an integration server. However, when an HTTP client attempts to call an operation that is not implemented, an `HTTP 501 Not Implemented` status code is returned to the client.

Procedure

To implement an operation in a REST API, complete the following steps:

1. Open the REST API that contains the operation that you want to implement.

   The **REST API Editor** is in the Application Development view under the REST API project.
2. Locate the operation in the **REST API Editor**. Operations are listed under the Operations heading, and are grouped by resource.
3. The **Header** section shows the details for the REST API base URL.
4. The **Resources** section shows the operations for each resource.

   Click this button to create a subflow for the operation:  . If you already defined a subflow for the operation, that subflow is opened. When you click this button, you are prompted to confirm that you want to save the REST API. Saving the RESTAPI ensures that the nodes that you add to the subflow are

correctly defined with the current definition of the REST API. The subflow is opened in the Message Flow Editor, and it is brought to the front.

Use the hover-help that is provided for each button to get information about the associated action.

5. Implement the operation by adding any of the standard IBM App Connect Enterprise message flow nodes to the subflow.

   For more information, see Implementing a REST API operation by using a message map.

6. Access the REST API operation information, REST API operation parameters, and REST API request and response body, as described in "Implementing REST API operation processing in the subflow by using message flow nodes" on page 2031.

Results

The operation is implemented in the REST API.

What to do next

You must package and deploy your REST API to an integration server, as described in "Packaging and deploying a REST API" on page 2045.

You can also complete the following optional tasks:

- Implement error handling for the REST API as a set of subflows. For more information, see " Implementing an error handler in a REST API" on page 2038.
- Secure your REST API by using HTTPS for encrypting communications between client and server. For more information, see "Securing a REST API by using HTTPS" on page 2041.
- Secure your REST API by authenticating users with HTTP Basic Authentication. For more information, see "Securing a REST API by using HTTP Basic Authentication" on page 2042.
- If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, as described in "Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing" on page 2043.
- You can enable JSON Validation for your REST API by using the "Message Validation" settings in the ACE REST API Editor. For more information about JSON Validation, see "JSON validation" on page 561.
- REST APIs are configured by default to handle JSON data. If you want to handle non-JSON data, see "Handling non-JSON data in a REST API" on page 2039.

Implementing REST API operation processing in the subflow by using message flow nodes

Access the REST API operation information, REST API operation parameters, and REST API request and response body.

Accessing the REST API operation information

Information about the current operation is automatically placed into the local environment tree. You can use this information in your implementation if you want to determine which operation in the REST API was called, which HTTP method was used, the request path, or the request URI. For example, if you are sharing transformation logic across multiple operations, you can use this information to determine in which operation the REST API was called.

Accessing current operation information in a message map

You can access the current operation information in a message map. The current operation information is available as a set of elements, which are named Method, Operation, Path, and URI, in a Message Assembly that includes the Local Environment, under **Local Environment** > **REST** > **Input folder**. For more information, see Mapping data in the local environment tree.

When you create the message map by using the option **Message map with input and output for REST API operation**, the local environment is automatically added to the map input and any path or query parameters are added. A Task transform is pre-wired to **Local Environment** > **REST** > **Input**,

to help you locate this information. For more information, see Implementing a REST API operation by using a message map.

Accessing current operation information in ESQL

Check to see if the current operation was called by using an HTTP GET.

```
IF InputLocalEnvironment.REST.Input.Method = 'GET' THEN
  -- Executed only if the current operation was called using an HTTP GET.
END IF;
```

Check to see if the current operation is the getAllCustomers operation.

```
IF InputLocalEnvironment.REST.Input.Operation = 'getAllCustomers' THEN
  -- Executed only if the current operation is the getAllCustomers operation.
END IF;
```

Create a log message with the request path.

```
DECLARE logMessagePath CHARACTER 'Received request on path ' ||
 InputLocalEnvironment.REST.Input.Path;
```

Create a log message with the request URI.

```
DECLARE logMessageURI CHARACTER 'Received request on URI  ' ||
 InputLocalEnvironment.REST.Input.URI;
```

Accessing current operation information in Java

Check to see if the current operation was called by using an HTTP GET.

```
MbMessage inLocalEnvironment = inAssembly.getLocalEnvironment();
MbElement leRestInput = inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/
Input");

if (leRestInput.getFirstElementByPath("Method").getValueAsString().equals("GET")) {
  // Executed only if the current operation was called using an HTTP GET.
}
```

Check to see if the current operation is the getAllCustomers operation.

```
if
 (leRestInput.getFirstElementByPath("Operation").getValueAsString().equals("getAllCustomers"))
 {
  // Executed only if the current operation is the getAllCustomers operation.
}
```

Create a log message with the request path.

```
String logMessagePath = "Received request on path " +
 leRestInput.getFirstElementByPath("Path").getValueAsString();
```

Create a log message with the request URI.

```
String logMessageURI = "Received request on URI  " +
 leRestInput.getFirstElementByPath("URI").getValueAsString();
```

Accessing current operation information in .NET (C#)

Check to see if the current operation was called by using an HTTP GET.

```
NBMessage inLocalEnvironment = inputAssembly.LocalEnvironment;
NBElement leRestInput = inLocalEnvironment.RootElement["REST"]["Input"];

if (((String) leRestInput["Method"]) == "GET"){
  // Executed only if the current operation was called using an HTTP GET.
}
```

Check to see if the current operation is the getAllCustomers operation.

```
if (((String) leRestInput["Operation"]) == "getAllCustomers") {
  //Executed only if the current operation is the getAllCustomers operation.
}
```

Create a log message with the request path.

```
String logMessagePath = "Received request on path " + ((String) leRestInput["Path"]);
```

Create a log message with the request URI.

```
String logMessageURI = "Received request on URI  " + ((String)leRestInput["URI"]);
```

Accessing the REST API operation parameters

If the definition of an operation includes one or more parameters, the names and values of those parameters are automatically placed into the local environment tree, but only if those parameters are provided by the HTTP client that is calling the operation. Optional or missing parameters are not placed into the local environment tree. Parameters are always placed into the local environment tree as character (string) elements, where the name of each element is the name of the parameter and the value of the element is the value of the parameter.

Accessing parameter values in a message map

You can access parameter values in a message map. All parameters are available as set of elements, where the elements have the same name as the parameter name, in a Message Assembly that includes the Local Environment, under **Local Environment** > **REST** > **Input** > **Parameters** folder.

When you create the message map by using the option **Message map with input and output for REST API operation**, the local environment is automatically added to the map input and any path or query parameters are added under **Local Environment** > **REST** > **Input** > **Parameters**, ready for you to connect transforms. A `Task` transform is pre-wired to **Local Environment** > **REST** > **Input** , to help you locate this information. For more information, see Implementing a REST API operation by using a message map. If you update the REST API and change the path or query parameters, the message map is updated with the new definitions when it is reopened. If there were existing transforms wired to the parameters, they might need to be modified.

If you do not use the option **Message map with input and output for REST API operation**, you must add the local environment manually, and in the **Parameters** folder use the `any` element to add user-defined elements. For more information, see Mapping data in the local environment tree and Defining user-defined elements.

Accessing parameter values in ESQL

Check to see if the parameter named 'max' is supplied.

```
DECLARE max INTEGER -1;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.max) IS NOT NULL THEN
  SET max = InputLocalEnvironment.REST.Input.Parameters.max;
END IF;
```

Accessing parameter values in Java

Check to see if the parameter named 'max' is supplied.

```
MbElement maxElement = inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/
Input/Parameters/max");
int max = -1;
if (maxElement != null) {
  max = Integer.valueOf(maxElement.getValueAsString());
}
```

Accessing parameter values in .NET (C#)

Check to see if the parameter named 'max' is supplied.

```
NBElement maxElement = inLocalEnvironment.RootElement["REST"]["Input"]["Parameters"]["max"];
int max = -1;
if (max != null){
  max = (int) maxElement;
}
```

Accessing the REST API request and response body

Depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. REST APIs in IBM App Connect Enterprise are configured by default to process JSON data. For more information about processing JSON data that was passed in the request body, and creating JSON data to send as the response body, see “JSON parser and domain” on page 553. You can also use a message map to process JSON data. For more information, see Creating or transforming a JSON output message by using a message map.

Implementing an operation for REST APIs based on OpenAPI 3.0 documents

Use the REST API Editor in the Application Development view or the **OpenAPI Editor** to implement operations in a REST API.

Before you begin

- Create a REST API based on an OpenAPI 3.0 specification document in the IBM App Connect Enterprise Toolkit, as described in “Creating a REST API” on page 2019.
- If you created the REST API from scratch, as described in “Creating a REST API from scratch with OpenAPI 3.0 specifications by using the IBM App Connect Enterprise Toolkit” on page 2022, you must also define the resources, models, and operations in the OpenAPI editor . For more information, see “Defining resources, models, and operations in a REST API by using the OpenAPI editor” on page 2029.

About this task

Operations in a REST API are implemented as a subflow, and you can use the RESTAPI Editor or the OpenAPI editor to create an empty subflow for each operation. You can then implement the operation by adding any of the standard IBM App Connect Enterprise message flow nodes to the subflow.

If you do not implement an operation, you can still deploy the REST API to an integration server. However, when an HTTP client attempts to call an operation that is not implemented, an `HTTP 501 Not Implemented` status code is returned to the client.

You can use the OpenAPI Editor to create, edit, and delete subflows for operations for OpenAPI 3.0 based REST APIs.

Procedure

To implement an operation in a REST API, complete the following steps:

1. Open the REST API Description by double-clicking the REST API project or by right-clicking and selecting **Open**.

   The REST API editor for OpenAPI 3.0 documents opens and provides a view of the API details, its Resources and Operations, Error handling, Security, and Message Validation Settings.
2. In the RESTPI Editor, open the REST API that contains the operation that you want to implement.
3. In the REST API Editor for OpenAPI 3.0, locate the operation in the **Resources and Operations** section and click **Create Subflow**. Alternatively, if you are working with the REST API in the OpenAPI editor , locate the operation and click **Create Subflow**.

   The new subflow is opened. You are prompted to confirm that you want to save the REST API. Saving the RESTAPI ensures that the nodes that you add to the subflow are correctly defined with the current

definition of the REST API. The subflow is opened in the Message Flow Editor, and it is brought to the front. If you already created a subflow, the actions **Edit Subflow** and **Delete Subflow** are provided instead of **Create Subflow**.

4. Implement the operation by adding any of the standard IBM App Connect Enterprise message flow nodes to the subflow.

   For more information, see Implementing a REST API operation by using a message map.

5. Access the REST API operation information, REST API operation parameters, and REST API request and response body, as described in "Implementing REST API operation processing in the subflow by using message flow nodes" on page 2031.

Results

The operation is implemented in the REST API.

What to do next

You must package and deploy your REST API to an integration server, as described in "Packaging and deploying a REST API" on page 2045.

You can also complete the following optional tasks:

- Implement error handling for the REST API as a set of subflows. For more information, see " Implementing an error handler in a REST API" on page 2038.
- Secure your REST API by using HTTPS for encrypting communications between client and server. For more information, see "Securing a REST API by using HTTPS" on page 2041.
- Secure your REST API by authenticating users with HTTP Basic Authentication. For more information, see "Securing a REST API by using HTTP Basic Authentication" on page 2042.
- If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, as described in "Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing" on page 2043.
- You can enable JSON Validation for your REST API by using the "Message Validation" settings in the ACE REST API Editor. For more information about JSON Validation, see "JSON validation" on page 561.
- REST APIs are configured by default to handle JSON data. If you want to handle non-JSON data, see "Handling non-JSON data in a REST API" on page 2039.

Implementing REST API operation processing in the subflow by using message flow nodes

Access the REST API operation information, REST API operation parameters, and REST API request and response body.

Accessing the REST API operation information

Information about the current operation is automatically placed into the local environment tree. You can use this information in your implementation if you want to determine which operation in the REST API was called, which HTTP method was used, the request path, or the request URI. For example, if you are sharing transformation logic across multiple operations, you can use this information to determine in which operation the REST API was called.

Accessing current operation information in a message map

You can access the current operation information in a message map. The current operation information is available as a set of elements, which are named Method, Operation, Path, and URI, in a Message Assembly that includes the Local Environment, under **Local Environment** > **REST** > **Input folder**. For more information, see Mapping data in the local environment tree.

When you create the message map by using the option **Message map with input and output for REST API operation**, the local environment is automatically added to the map input and any path or query parameters are added. A Task transform is pre-wired to **Local Environment** > **REST** > **Input**,

to help you locate this information. For more information, see Implementing a REST API operation by using a message map.

Accessing current operation information in ESQL

Check to see if the current operation was called by using an HTTP GET.

```
IF InputLocalEnvironment.REST.Input.Method = 'GET' THEN
  -- Executed only if the current operation was called using an HTTP GET.
END IF;
```

Check to see if the current operation is the getAllCustomers operation.

```
IF InputLocalEnvironment.REST.Input.Operation = 'getAllCustomers' THEN
  -- Executed only if the current operation is the getAllCustomers operation.
END IF;
```

Create a log message with the request path.

```
DECLARE logMessagePath CHARACTER 'Received request on path ' ||
 InputLocalEnvironment.REST.Input.Path;
```

Create a log message with the request URI.

```
DECLARE logMessageURI CHARACTER 'Received request on URI  ' ||
 InputLocalEnvironment.REST.Input.URI;
```

Accessing current operation information in Java

Check to see if the current operation was called by using an HTTP GET.

```
MbMessage inLocalEnvironment = inAssembly.getLocalEnvironment();
MbElement leRestInput = inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/
Input");

if (leRestInput.getFirstElementByPath("Method").getValueAsString().equals("GET")) {
  // Executed only if the current operation was called using an HTTP GET.
}
```

Check to see if the current operation is the getAllCustomers operation.

```
if
 (leRestInput.getFirstElementByPath("Operation").getValueAsString().equals("getAllCustomers"))
 {
  // Executed only if the current operation is the getAllCustomers operation.
}
```

Create a log message with the request path.

```
String logMessagePath = "Received request on path " +
 leRestInput.getFirstElementByPath("Path").getValueAsString();
```

Create a log message with the request URI.

```
String logMessageURI = "Received request on URI  " +
 leRestInput.getFirstElementByPath("URI").getValueAsString();
```

Accessing current operation information in .NET (C#)

Check to see if the current operation was called by using an HTTP GET.

```
NBMessage inLocalEnvironment = inputAssembly.LocalEnvironment;
NBElement leRestInput = inLocalEnvironment.RootElement["REST"]["Input"];

if (((String) leRestInput["Method"]) == "GET"){
  // Executed only if the current operation was called using an HTTP GET.
}
```

Check to see if the current operation is the getAllCustomers operation.

```
if (((String) leRestInput["Operation"]) == "getAllCustomers") {
  //Executed only if the current operation is the getAllCustomers operation.
}
```

Create a log message with the request path.

```
String logMessagePath = "Received request on path " + ((String) leRestInput["Path"]);
```

Create a log message with the request URI.

```
String logMessageURI = "Received request on URI  " + ((String)leRestInput["URI"]);
```

Accessing the REST API operation parameters

If the definition of an operation includes one or more parameters, the names and values of those parameters are automatically placed into the local environment tree, but only if those parameters are provided by the HTTP client that is calling the operation. Optional or missing parameters are not placed into the local environment tree. Parameters are always placed into the local environment tree as character (string) elements, where the name of each element is the name of the parameter and the value of the element is the value of the parameter.

Accessing parameter values in a message map

You can access parameter values in a message map. All parameters are available as set of elements, where the elements have the same name as the parameter name, in a Message Assembly that includes the Local Environment, under **Local Environment** > **REST** > **Input** > **Parameters** folder.

When you create the message map by using the option **Message map with input and output for REST API operation**, the local environment is automatically added to the map input and any path or query parameters are added under **Local Environment** > **REST** > **Input** > **Parameters**, ready for you to connect transforms. A `Task` transform is pre-wired to **Local Environment** > **REST** > **Input** , to help you locate this information. For more information, see Implementing a REST API operation by using a message map. If you update the REST API and change the path or query parameters, the message map is updated with the new definitions when it is reopened. If there were existing transforms wired to the parameters, they might need to be modified.

If you do not use the option **Message map with input and output for REST API operation**, you must add the local environment manually, and in the **Parameters** folder use the `any` element to add user-defined elements. For more information, see Mapping data in the local environment tree and Defining user-defined elements.

Accessing parameter values in ESQL

Check to see if the parameter named 'max' is supplied.

```
DECLARE max INTEGER -1;
IF FIELDTYPE(InputLocalEnvironment.REST.Input.Parameters.max) IS NOT NULL THEN
  SET max = InputLocalEnvironment.REST.Input.Parameters.max;
END IF;
```

Accessing parameter values in Java

Check to see if the parameter named 'max' is supplied.

```
MbElement maxElement = inLocalEnvironment.getRootElement().getFirstElementByPath("/REST/
Input/Parameters/max");
int max = -1;
if (maxElement != null) {
  max = Integer.valueOf(maxElement.getValueAsString());
}
```

Accessing parameter values in .NET (C#)

Check to see if the parameter named 'max' is supplied.

```
NBElement maxElement = inLocalEnvironment.RootElement["REST"]["Input"]["Parameters"]["max"];
int max = -1;
if (max != null){
  max = (int) maxElement;
}
```

Accessing the REST API request and response body

Depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. REST APIs in IBM App Connect Enterprise are configured by default to process JSON data. For more information about processing JSON data that was passed in the request body, and creating JSON data to send as the response body, see “JSON parser and domain” on page 553. You can also use a message map to process JSON data. For more information, see Creating or transforming a JSON output message by using a message map.

Implementing an error handler in a REST API

Use the REST API Description view to implement error handlers in a REST API.

Before you begin

You must create a REST API in the REST API Description, see “Creating a REST API” on page 2019.

About this task

Three types of error handler are available for a REST API:

Catch
If an exception is thrown when you process a request in a subflow for an operation and that exception is not handled by that subflow, a message is routed to the Catch error handler. For example, if a JavaCompute node in the subflow for an operation throws an exception and that exception is not caught by a TryCatch node, that exception is passed to the Catch error handler.

Timeout
If a subflow for an operation is processing a message and that subflow does not respond to the client within the expected time limit, a message is routed to the Timeout error handler. The Timeout error handler can then be used to pass a response back to the client to inform that client that the operation timed out. If this situation occurs, the subflow for an operation continues to process the message. The processing of that message is not cancelled. However, a response to the client cannot be sent.

Note: Depending on the client, the client might time out without receiving a response from calling the operation, in which case, any response that is sent is not received.

Failure
If an error occurs while the request from the client is being processed and that error is not handled by the Catch or Timeout error handlers, a message is routed to the Failure error handler.

Error handlers in a REST API are implemented as a subflow. You must use the IBM App Connect Enterprise Toolkit to create an empty subflow for each error handler. You can then implement the error handler by adding any of the standard message flow nodes that are available in IBM App Connect Enterprise to the subflow.

If you do not implement an error handler, default error handling behavior is used. If you do not implement the Catch error handler or the Failure error handler, the exception is thrown back to the client with an HTTP 500 Internal Server Error status code. If you do not implement the Timeout error handler, a timeout exception is thrown back to the client with an HTTP 504 Gateway Timeout status code.

Procedure

To implement an error handler in a REST API, complete the following steps:

1. Open the `REST API Description` for the REST API for which you want to implement an error handler.

   The `REST API Description` is in the Application Development view under the REST API project.
2. Locate the error handler in the `REST API Description`.

   Error handlers are listed under the Error Handling heading.

Complete one of the following steps depending on whether you created the RESTAPI with a Swagger document or an OpenAPI3 document.

3. Optional: If you created the RESTAPI with a Swagger document, click the **Implement the *<error_handler_type>*** error handler link, where *<error_handler_type>* is the error handler type.

   A new subflow is automatically created and opened. The link on the `REST API Description` changes to **Open the <error_handler_type>** error handler, and if you click the link, the existing subflow reopens.
4. Optional: If you created the REST API with an OpenAPI 3.0 document, click **Create subflow** to create a subflow.

   You can also use the OpenAPI Editor **Configuration** tab to implement error handlers for REST APIs based on OpenAPI 3.0.

Results

The error handler is implemented in the REST API.

What to do next

You must package and deploy your REST API to an integration server, see “Packaging and deploying a REST API” on page 2045.

You can also complete the following optional tasks:

- Use the `REST API Description` view to implement operations in a REST API, see “Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents” on page 2030 or “Implementing an operation for REST APIs based on OpenAPI 3.0 documents” on page 2034.
- Secure your REST API by using HTTPS for encrypting communications between client and server, see “Securing a REST API by using HTTPS” on page 2041.
- Secure your REST API by authenticating users with HTTP Basic Authentication, see “Securing a REST API by using HTTP Basic Authentication” on page 2042.
- If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, see “Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing” on page 2043.
- You can enable JSON Validation for your REST API by using the "Message Validation" settings in the ACE REST API Editor. For more information about JSON Validation, see “JSON validation” on page 561.
- REST APIs are configured by default to handle JSON data. If you want to handle non-JSON data, see “Handling non-JSON data in a REST API” on page 2039.

Handling non-JSON data in a REST API

REST APIs are set up by default to handle JSON data, but can also handle other data formats.

Before you begin

You must complete the following tasks:

1. You must create a REST API in the IBM App Connect Enterprise Toolkit, see “Creating a REST API” on page 2019.

2. You must implement at least one operation in the REST API, see “Implementing an operation in a REST API by using the REST API Editor for Swagger 2 documents” on page 2030.

About this task

When a message arrives on the Input node of a subflow for an implemented operation, the message domain of that message is set to JSON. You can handle other types of data in a subflow for an implemented operation by using a ResetContentDescriptor node to change the message domain of the message before any message parsing is performed.

You can also use a Route node to handle multiple types of data in the same subflow. You can configure the Route node to route data to a ResetContentDescriptor node based on the Content-Type header in the request message. The Content-Type header can be added to the request by the HTTP client, and is used to describe the type of the data in the request.

Procedure

To handle non-JSON data in a REST API, complete the following steps:

1. Open the `REST API Description`, for the implemented operation that you want to modify, click the **Open the Operation** link.
2. Optional: If you want to handle multiple types of data in this subflow, complete the following steps:
   a) Add a Route node to the canvas.
   b) Connect the **Out** terminal of the subflow Input node to the **In** terminal of the Route node.
   c) Add Output terminals to the Route node for each message domain and type of data that you want to process by right-clicking the Route node and clicking **Add Output Terminal**.
   d) Route to the new terminals on the Route node by adding entries to the **Filter** table.
   e) To route on Content-Type, use the following XPath expression as an example for handling XML request messages by using the XMLNSC domain:

   ```
   starts-with($Root/HTTPInputHeader/Content-Type,'application/xml')
   ```

   Use the starts-with function in your XPath expression because HTTP clients might send in an optional character set parameter following the type and subtype.
3. Add a ResetContentDescriptor node to the canvas.
   - If you added a Route node in the previous step, you must add a ResetContentDescriptor node for each message domain and type of data that you want to process. If you are processing JSON data, you do not require a ResetContentDescriptor node.
   - If you did not add a Route node in the previous step, connect the **Out** terminal of the subflow Input node to the **In** terminal of the ResetContentDescriptor node.
   - If you added a Route node in the previous step, then connect the terminal that you added in the previous step to the corresponding ResetContentDescriptor node for that message domain and type of data.
4. For each ResetContentDescriptor node that you added in the previous step, configure the ResetContentDescriptor node by specifying the value of the **Message domain** property. You must also select **Reset message domain**.
5. You can now continue to implement the operation by using any of the standard message flow nodes that are included in IBM App Connect Enterprise. Connect any new message flow nodes to the **Out** terminal of the IBM App Connect Enterprise nodes that you added in the previous steps.

Results

Your REST API is set up to handle non-JSON data.

What to do next

You must package and deploy your REST API to an integration server, see "Packaging and deploying a REST API" on page 2045.

You can also complete the following optional tasks:

- Implement error handling for the REST API as a set of subflows, see " Implementing an error handler in a REST API" on page 2038.
- Secure your REST API by using HTTPS for encrypting communications between the client and server, see "Securing a REST API by using HTTPS" on page 2041.
- Secure your REST API by authenticating users with HTTP Basic Authentication, see "Securing a REST API by using HTTP Basic Authentication" on page 2042.
- If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, see "Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing" on page 2043.

Securing a REST API by using HTTPS

Secure the communications between a REST API and an HTTP client by enabling HTTPS.

Before you begin

- Create a REST API in the IBM App Connect Enterprise Toolkit. Follow the instructions in "Creating a REST API" on page 2019. This makes the REST API available to be configured for HTTPS.
- Create the integration server to which you want to deploy the REST API. Follow the instructions in "Creating an integration server" on page 168.
- Decide which HTTP Listener you want to use for HTTPS messages. For information about which listener to use for HTTPS messages, see "HTTP listeners" on page 795.
- Set up a public key infrastructure (PKI) to configure the keystores, truststores, passwords, and certificates to enable SSL communication. Follow the instructions in "Setting up a public key infrastructure" on page 2635. This results in the integration server or integration node being configured for the PKI.

About this task

Secure the communications between a REST API and an HTTP client by enabling HTTPS. You can enable HTTPS just for encryption, or you can also configure a REST API for client authentication (mutual authentication).

This task uses some of the same substeps as enabling a message flow with HTTPInput and HTTPReply nodes to use HTTPS, as described in "Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 2619.

Procedure

To enable HTTPS for a REST API, complete the following steps:

1. Configure the integration server or integration node to use SSL.

   Complete one of the following substeps, depending on which HTTP listener you have chosen to use for HTTPS messages:

   - If you are using the integration node listener: Configure the integration node to use SSL
   - If you are using the integration server listener: Configure the integration server to use SSL
2. In the Application Development view, which is under the REST API project, open the REST API `Description` for the REST API for which you want to enable HTTPS.
3. Under Security Options, select **Enable HTTPS** in the REST API Description.

Results

Your REST API is secured by using HTTPS.

What to do next

1. You can complete the following optional tasks:
   - Secure your REST API by authenticating users with HTTP Basic Authentication, see "Securing a REST API by using HTTP Basic Authentication" on page 2042.
   - If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, see "Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing" on page 2043.
2. Package and deploy your REST API to an integration server, see "Packaging and deploying a REST API" on page 2045.

Securing a REST API by using HTTP Basic Authentication

Authenticate HTTP clients that want to call a REST API by enabling HTTP Basic Authentication.

Before you begin

You must create a REST API in the IBM App Connect Enterprise Toolkit, see "Creating a REST API" on page 2019.

About this task

You can authenticate HTTP clients that want to call a REST API by enabling HTTP Basic Authentication. IBM App Connect Enterprise supports several authentication providers that can be used for this purpose, including Lightweight Directory Access Protocol (LDAP), any WS-Trust V1.3 compliant Security Token Service (STS), and Tivoli Federated Identity Manager.

Procedure

To enable HTTP Basic Authentication for a REST API by using Lightweight Directory Access Protocol (LDAP), any WS-Trust V1.3 compliant Security Token Service (STS), or Tivoli Federated Identity Manager, complete the following steps:

1. Create a security profile that you can use for authentication, see "Creating a security profile for LDAP" on page 2584, "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588 and "Creating a security profile for TFIM V6.1" on page 2589.
2. Configure the security profile that you created in the previous step on the REST API:
   a) In the **BAR file** editor, open the BAR file that contains the REST API.
   b) Click the **Manage** tab.
   c) Locate the automatically generated message flow in the REST API.
      The automatically generated message flow is named `gen/<name_of_REST API>`, where *<name_of_REST API>* is the name of the REST API.
   d) In the **Properties** view for the automatically generated message flow, in the **Security Profile Name** field, specify the name of the security profile that you created in the previous step. Save the BAR file.
3. Deploy the BAR file to an integration server.
   HTTP clients that want to call the REST API must provide a user name and password. If authentication information is not supplied by the HTTP client, or authentication fails, an `HTTP 401 Unauthorized` status code is returned to the client.

Results

You have secured a REST API by using HTTP Basic Authentication.

What to do next

You must package and deploy your REST API to an integration server, see "Packaging and deploying a REST API" on page 2045.

You can also complete the following optional tasks:

- Secure your REST API by using HTTPS, see "Securing a REST API by using HTTPS" on page 2041.
- If your REST API is going to be used by client-side code that is running in a web browser, you might have to configure Cross-Origin Resource Sharing, see "Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing" on page 2043.

Permitting web browsers to access a REST API by using Cross-Origin Resource Sharing

Permit web pages that are running in a web browser to make requests to a REST API that is running in IBM App Connect Enterprise by enabling Cross-Origin Resource Sharing (CORS).

Before you begin

You must create a REST API in the IBM App Connect Enterprise Toolkit, see "Creating a REST API" on page 2019.

About this task

You can permit a web browser to access a REST API by using CORS. When you enable CORS on an integration server, it is enabled for all REST APIs and any other HTTP services that are running on that integration server. You are not required to configure CORS for each REST API that you deploy.

Procedure

To permit a web browser to access a REST API, complete the following steps:

1. Configure the integration server HTTP listener to enable CORS.
2. Ensure that the CORS configuration meets the requirements for all operations that are deployed in the REST API.

   To permit cross-origin requests for additional HTTP methods, additional HTTP headers, or to allow authentication information to be passed into the REST API, you might have to change some extra parameters.

Results

Your web browser can access a REST API by using CORS.

What to do next

You must package and deploy your REST API to an integration server, see "Packaging and deploying a REST API" on page 2045.

You can also complete the following optional tasks:

- Secure your REST API by using HTTPS for encrypting communications between client and server, see "Securing a REST API by using HTTPS" on page 2041.
- Secure your REST API by authenticating users with HTTP Basic Authentication, see "Securing a REST API by using HTTP Basic Authentication" on page 2042.

Creating a new REST API that uses artifacts from an existing REST API

Deploy multiple versions of the same REST API simultaneously by copying REST API artifacts into a new REST API project.

Before you begin

You must create a REST API in the IBM App Connect Enterprise Toolkit, see "Creating a REST API" on page 2019.

About this task

To create multiple versions of the same REST API, you can copy artifacts from an existing REST API into a new REST API.

Note: If you want to deploy the new and existing REST APIs on the same integration server, you must specify a different base path for each of them. Use the Modify button () in the REST API editor to modify the resource paths.

Procedure

To create a new REST API that uses artifacts from an existing REST API complete the following steps.

1. Create a REST API project in the IBM App Connect Enterprise Toolkit.

   If the existing REST API project is in the currently open workspace, you must specify a different name for the new REST API project. If you modified the REST API project, modify the resource path by using the Modify button in the REST API editor.
2. Open the `REST API Description` for the new REST API project.

Complete one of the following steps depending on whether you created the REST API with a Swagger document or an OpenAPI 3.0 document.

3. Optional: In the `REST API Description`, implement operations and error handlers from the existing REST API project.
   a) for all operations that you want to implement, click the **Implement the Operation** link.
   b) If you created the REST API with a Swagger document, then for all error handlers that you want to implement, click the **Implement the Error Handler** link.

   This action creates the connections from the REST API project to the subflows.

   Note: Do not add any message flow nodes to the generated subflows.
4. Optional: In the `REST API Description`, implement operations and error handlers from the existing REST API project.
   a) If you created the REST API with an OpenAPI 3.0 document, then for all operations that you want to implement, click the **Create subflow** button.
   b) If you created the REST API with an OpenAPI 3.0 document, then, for all error handlers that you want to implement, click the **Create subflow** button.

   This action creates the connections from the REST API project to the subflows.

   Note: Do not add any message flow nodes to the generated subflows.
5. Copy the subflows for implemented operations and error handlers from the existing REST API project into the new REST API project.

   Copying subflows from the existing REST API overwrites the subflows that are generated in the previous step.
6. Reference any static or shared libraries in the new REST API project that are referenced by the existing REST API, see "Referencing resources in other libraries" on page 1976.

7. Copy all other artifacts, for example, subflows, schemas, and maps, from the existing REST API project into the new REST API project.

Results

You have created a new REST API by reusing artifacts from an existing REST API.

What to do next

You must package and deploy your REST API to an integration server, see "Packaging and deploying a REST API" on page 2045.

Packaging and deploying a REST API

Package your REST API into a BAR file and deploy it to an integration server.

Before you begin

You can create a REST API in the IBM App Connect Enterprise Toolkit, as described in "Creating a REST API" on page 2019.

About this task

REST APIs can be deployed to integration servers that are configured to use the integration server HTTP listener.

If you plan to deploy more than one REST API to an integration server, the REST APIs must have different base paths.

Procedure

To package a REST API into a BAR file and deploy it to an integration server, complete the following steps:

1. Package the REST API into a BAR file.

   You can use either the **BAR file editor**, or you can use the command line. Both the **mqsicreatebar** and **mqsipackagebar** commands package a REST API into a BAR file. When you use these commands, pass the name of the REST API in as the application name.

   For example:

   ```
   mqsicreatebar -data workspaceDirectory -b restApiName.bar -a restApiName
   ```

   ```
   mqsipackagebar -a restApiName.bar -k restApiName
   ```

2. Deploy the REST API to the integration server.

   You can use the IBM App Connect Enterprise Toolkit, the web user interface, or the **mqsideploy** command; for example:

   ```
   mqsideploy integrationNodeName -e integrationServerName -a restApiName.bar
   ```

   ```
   mqsideploy -i ipAddress -p port -a C:\myworkspace\myapi1\restApiName.bar
   ```

   If another REST API is deployed to the integration server and that REST API has a clashing base path, the deployment of the REST API fails. You cannot deploy REST APIs that have clashing base paths to the same integration server.

Results

Your REST API is deployed.

Pushing REST APIs to IBM API Connect

Create or update a definition for your deployed REST APIs on IBM API Connect.

Before you begin

You must complete the following tasks:

- Register or update credentials for the API owner with IBM API Connect:
  - For Version 5 and Version 5 on IBM Cloud, see Administering Developer organizations in the IBM API Connect product documentation.
  - For Version 2018, see Administering Consumer organizations in the IBM API Connect product documentation.
  - For Version 10, see Working with consumer organizations in the IBM API Connect product documentation.
- If you intend to stage the REST API to a catalog, ensure that the IBM API Connect server has a catalog to which you can stage a product that includes the REST API.
- Ensure that any firewalls allow the following connections:
  - IBM App Connect Enterprise can connect to the IBM API Connect server by using HTTPS.
  - The IBM API Connect server can connect to the integration server by using HTTP or HTTPS.
- Create a REST API in the IBM App Connect Enterprise Toolkit; see “Creating a REST API” on page 2019.
- Deploy the REST API to the integration server; see “Packaging and deploying a REST API” on page 2045.

About this task

You can push the definition of deployed REST APIs into IBM API Connect, which you can then use to manage and publish your APIs. Use IBM API Connect to promote and monitor the usage of the REST APIs, and to secure and authenticate access requests from users and applications to your IBM App Connect Enterprise application. It also provides monitoring and fine-grained control over access and workload management, such as the ability to control the number of requests per second for a client application. For information about supported versions of IBM API Connect, see the IBM App Connect Enterprise system requirements web page.

IBM API Connect works with an IBM DataPower® Gateway appliance. IBM API Connect pushes configuration to the IBM DataPower Gateway, which in turn proxies the requests back to the IBM App Connect Enterprise integration server that hosts the deployed REST API, as shown in the following diagram.

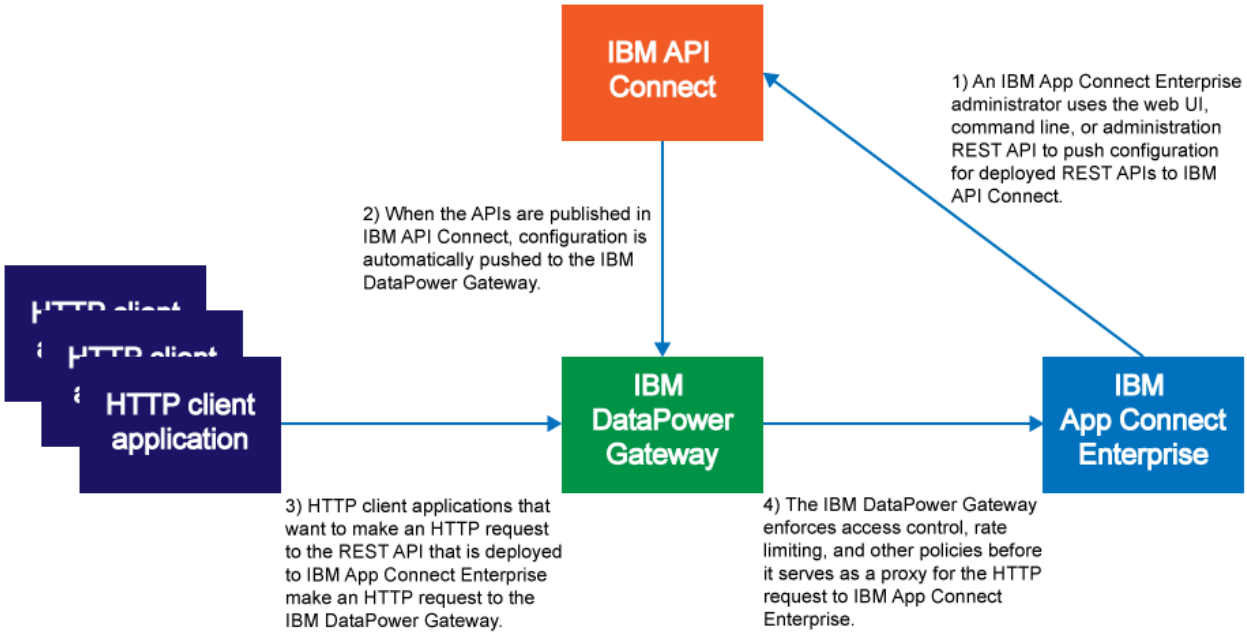


Figure 35. The interaction between IBM App Connect Enterprise, IBM API Connect, IBM DataPower Gateway, and HTTP client applications.

When you push a REST API to IBM API Connect, you can either create a new REST API definition or, if the REST API definition exists on the IBM API Connect system, you can replace the latest version of the definition with the new definition of your REST API. The title and version of the REST API are used to identify the REST API definition on the IBM API Connect system.

You can push one or more deployed REST APIs to IBM API Connect by using the IBM App Connect Enterprise web user interface, the **mqsipushapis** command, or the administration REST API, as described in the following topics:

- "Pushing REST APIs to IBM API Connect by using the web user interface" on page 2047
- "Pushing REST APIs to IBM API Connect by using the mqsipushapis command" on page 2052

Pushing REST APIs to IBM API Connect by using the web user interface

Use the web user interface to create or update a definition for one or more deployed REST APIs in IBM API Connect.

Before you begin

Complete the following tasks:

- Register or update credentials for the API owner with IBM API Connect:
  - For Version 5 and Version 5 on IBM Cloud, see Administering Developer organizations in the IBM API Connect product documentation.
  - For Version 2018, see Administering Consumer organizations in the IBM API Connect product documentation.
  - For Version 10, see Working with consumer organizations in the IBM API Connect product documentation.
- If you intend to stage the REST API to a catalog, ensure that the IBM API Connect server has a catalog to which you can stage a product that includes the REST API. If a current version of the REST API exists in the catalog, staging a new version of the REST API unpublishes the current version and changes it to staged state.

- Ensure that any firewalls allow the following connections:
  - IBM App Connect Enterprise can connect to the IBM API Connect server by using HTTPS.
  - The IBM API Connect server can connect to the integration server by using HTTP or HTTPS.
- Create a REST API in the IBM App Connect Enterprise Toolkit; see "Creating a REST API" on page 2019.
- Deploy the REST API to the integration server; see "Packaging and deploying a REST API" on page 2045.

About this task

You can use the IBM App Connect Enterprise web user interface to push one or more REST APIs to IBM API Connect. If the REST API definition exists on the IBM API Connect system, you replace the REST API definition with the new definition of your REST API.

Procedure

To push one or more REST APIs to IBM API Connect, complete the following steps.

1. In the web user interface, click the **Open List of Options** icon for the required integration server.
   - If you are viewing all the available servers under the **Servers** tab, the **Open List of Options** icon appears within the tile for each integration server.
   - If you are viewing the content of an integration server under the **Contents** tab, the **Open List of Options** icon appears in the title bar for the integration server.
2. Click **Share REST APIs to API Connect**.

   The **Push REST APIs to API Connect** dialog box opens for you to define your connection to the IBM API Connect system.
3. Select the version of the IBM API Connect system that you are connecting to.

   You can choose from Version 5, Version 5 on IBM Cloud, Version 2018, or Version 10.
4. Enter the connection details for the IBM API Connect management cluster or server in the **Host** and **Port** fields.
   - For **Version 5**, enter the values of the host and port for the IBM API Connect management server or cluster.
   - For **Version 5 on IBM Cloud**, enter the values of the host and port for the IBM API Connect management server or cluster:
     a. Enter the URL of the IBM API Connect system in the **Host** field, omitting the `http://` or `https://` protocol but including the region code; for example:

     ```
     apimanager.eu-gb.apiconnect.cloud.ibm.com
     ```

     In this example, `eu-gb` is the region code for London. The regions that you can specify are:
     - Sydney: `au-syd`
     - Frankfurt: `eu-de`
     - London:`eu-gb`
     - Tokyo: `jp-tok`
     - Washington DC:`us-east`
     - Dallas: `us-south`

     For more information about region codes, see Updated global location names for IBM Cloud.
     b. Enter the port number in the **Port** field; for example, 443.
   - For **Version 2018** or **Version 10**, enter the values of the host and port for the IBM API Connect management server or cluster.

5. Enter your credentials for accessing the IBM API Connect system.

   Note:

   If the IBM API Connect environment is configured with self-signed certificates, or if the certificate configuration does not match the domain of the IBM API Connect environment, the first stage of the **Push REST APIs to API Connect** process fails. If you select **Disable certificate verification when connecting**, App Connect Enterprise ignores the errors. This property is not required when you connect to IBM API Connect Version 5 on IBM Cloud.

   - For **Version 5**, enter your username and password.
   - For **Version 5 on IBM Cloud**, enter the API key that is specified in your service credentials in IBM API Connect. To generate an API key, complete these steps:
     a. Open the IBM Cloud Dashboard at https://cloud.ibm.com and log in if necessary.
     b. From the menu, click **Manage** > **Access (IAM)**.
     c. In the Access (IAM) contents pane, click **IBM Cloud API Keys**.
     d. Click **Create an IBM Cloud API Key**, then follow the on-screen instructions.
   - For **Version 2018** or **Version 10**, enter your username and password, together with the realm, client ID, and client secret.

     You can obtain the values for the client ID and client secret by registering App Connect Enterprise as a client application that can access the IBM API Connect environment. To register a client app from the IBM API Connect toolkit command-line interface (CLI), complete the following steps:

     a. Create a file called `ace-registration.json`, which will be used as the input for creating a registration object. Add the following JSON object as the file content, where *clientID* and *clientSecret* represent appropriate secure values of your choice.

```
{
  "name": "ace-registration",
  "client_id": "clientID",
  "client_secret": "clientSecret",
  "client_type": "toolkit"
}
```

     Example:

```
{
  "name": "ace-registration",
  "client_id": "aceappid4321",
  "client_secret": "aceapptopsecret",
  "client_type": "toolkit"
```

```
}
```

b. If you don't already have a local installation of the IBM API Connect toolkit that provides CLI commands for IBM API Connect, download and install the toolkit. For more information, see Installing the toolkit in the IBM API Connect documentation.

c. From a command prompt, log in to the IBM API Connect management server by using either of the following toolkit CLI commands. Ensure that you log in as an admin with the authority to create registrations.

Interactive login:

```
apic-slim login
```

(You will be prompted to enter your IBM API Connect credentials.)

Non-interactive login:

```
apic-slim login --username userID --password userPassword --server mgmtServerEndpoint
 --realm realm
```

For example:
`apic-slim login --username joebloggs --password mysecretpwd --server mycloudmanager.mydomain.com --realm admin/default-idp-1`

For more information about these login credentials, see Logging in to a management server in the IBM API Connect documentation.

d. Create an IBM API Connect registration for App Connect Enterprise by running the following command, which supplies the `ace-registration.json` file that you created earlier, as input.

```
apic-slim registrations:create --server mgmtServerEndpoint ace-registration.json
```

If you specify an existing `client_id` value in the `ace-registration.json` file, the registration process silently generates a new client ID. You must therefore verify the `client_id`

value for the registration before you specify it in the **Push REST APIs to API Connect** dialog box. (See the next step for details.)

e. Confirm the registration by running the following command:

```
apic-slim registrations:get ace-registration --server mgmtServerEndpoint --output -
```

For example:
`apic-slim registrations:get ace-registration --server mycloudmanager.mydomain.com --output -`

You should see output similar to this:

```
type: registration
api_version: 2.0.0
id: 6a45ed7e-d461-4ea7-89fa-f2a84b2f28ed
name: ace-registration
title: ace-registration
state: enabled
client_type: toolkit
client_id: d2ddd06b-679b-4e84-a7db-c3a18aa3da52
client_secret: '********'
scopes:
  - 'cloud:view'
  - 'cloud:manage'
  - 'provider-org:view'
  - 'provider-org:manage'
  - 'org:view'
  - 'org:manage'
  - 'product-drafts:view'
  - 'product-drafts:edit'
  - 'api-drafts:view'
  - 'api-drafts:edit'
  - 'child:view'
  - 'child:create'
  - 'child:manage'
  - 'product:view'
  - 'product:stage'
  - 'product:manage'
  - 'approval:view'
  - 'approval:manage'
  - 'api-analytics:view'
  - 'api-analytics:manage'
  - 'consumer-org:view'
  - 'consumer-org:manage'
  - 'app:view:all'
  - 'app:manage:all'
  - 'my:view'
  - 'my:manage'
  - 'webhook:view'
created_at: '2021-01-25T14:38:49.000Z'
updated_at: '2021-01-25T14:38:49.000Z'
url: >-
  https://mycloudmanager.mydomain.com/api/cloud/registrations/6a45ed7e-d461-4ea7-89fa-
f2a84b2f28ed
```

If the `client_id` value in the output is different from the value that you specified in the `ace-registration.json` file, this indicates that a new client ID was generated by the registration process. Make a note of this new `client_id` value because it takes precedence over the value in your `ace-registration.json` file.

Tip: You can alternatively confirm the registration by running this command, which writes the output to a file named `ace-registration.yaml`:

```
apic-slim registrations:get ace-registration --server mgmtServerEndpoint
```

f. In the **Push REST APIs to API Connect** dialog box, enter the `client_id` value (from the **`apic-slim registrations:get`** output) into the **Client ID** field, and enter the `client_secret` value from the `ace-registration.json` file into the **Client secret** field.

For more information about creating a JSON object for the `ace-registration.json` file and a client registration, see Obtaining a Client ID and Secret in the Open API Explorer Documentation.

6. After you enter your credentials, click **Connect to API Connect**.

   A connection is established to the IBM API Connect server and you are prompted to specify the target organization, product, and catalog.
7. Select the target organization where you want to push the REST APIs.

   The **Organization** menu shows a list of organizations on the IBM API Connect server to which you have access.
8. Specify the name of the product that you want to update or create, then click **Continue**.

   You can specify an existing product or you can create a new product. Specify the title, name, and version of the product. If you want to stage the product, specify the name of the catalog in which you want it to be staged.

   Available REST APIs are listed.
9. Select the REST APIs that you want to push to IBM API Connect and add to the product, then click **Continue**.

   You are given the option to specify the endpoint that IBM API Connect uses to call the REST API that is hosted by App Connect Enterprise.
10. Choose to either override the hostname and port number that are used by IBM API Connect to call the pushed APIs, or allow IBM API Connect to call the APIs directly.

    If you choose to override the hostname and port number, you can specify HTTP proxy details, HTTPS proxy details, or both.

    - To complete the process of pushing the REST APIs to IBM API Connect by using the web user interface, click **Push APIs**.
    - Optionally, you can copy the properties that you set through the web user interface to the clipboard, in command format, by clicking **Copy command**. You can then use this output to modify and run the **mqsipushapis** command to push the REST APIs to IBM API Connect. For more information, see "Pushing REST APIs to IBM API Connect by using the mqsipushapis command" on page 2052.
11. You can view progress as the selected REST APIs are pushed to IBM API Connect.

    - If an API definition exists on the IBM API Connect server, the existing definition is replaced by the definition of your REST API. If no API definition exists on the server, a new API definition is created for your REST API.
    - If the specified product exists, it is updated. If the specified product does not exist, a new product is created.
    - If a catalog is selected, the product is staged in the catalog.
12. Click individual tasks to see a detailed breakdown of the progress of each task.

    When the REST APIs are pushed to IBM API Connect, close the wizard by clicking **Done**.

Results

Your REST API definition is created or updated on the IBM API Connect server.

Pushing REST APIs to IBM API Connect by using the `mqsipushapis` command

Use the **mqsipushapis** command to create or update the definition of one or more deployed REST APIs on IBM API Connect.

Before you begin

You must complete the following tasks:

- Register or update credentials for the API owner with IBM API Connect:
  - For Version 5 and Version 5 on IBM Cloud, see Administering Developer organizations in the IBM API Connect product documentation.
  - For Version 2018, see Administering Consumer organizations in the IBM API Connect product documentation.
  - For Version 10, see Working with consumer organizations in the IBM API Connect product documentation.
- If you intend to stage the REST API to a catalog, ensure that the IBM API Connect server has a catalog to which you can stage a product that includes the REST API. If a current version of the REST API exists in the catalog, then staging a new version of the REST API unpublishes the current version and changes it to staged state.
- Ensure that any firewalls allow the following connections:
  - IBM App Connect Enterprise can connect to the IBM API Connect server by using HTTPS.
  - The IBM API Connect server can connect to the integration server by using HTTP or HTTPS.
- Create a REST API in the IBM App Connect Enterprise Toolkit; see “Creating a REST API” on page 2019.
- Deploy the REST API to the integration server; see “Packaging and deploying a REST API” on page 2045.

About this task

You can use the **mqsipushapis** command to push the definition of one or more REST APIs to IBM API Connect. If the REST API definition exists on the IBM API Connect system, you replace the REST API definition with the new definition of your REST API.

Procedure

Follow these steps to push one or more deployed REST APIs to the IBM API Connect server:

1. Optional: Construct the **mqsipushapis** command in the web user interface by using the **Push REST APIs to API Connect** dialog and then clicking **Copy command** to copy the output to the clipboard. For more information, see step “1” on page 2048 to step “10” on page 2052 in “Pushing REST APIs to IBM API Connect by using the web user interface” on page 2047. You can use the output to modify the **mqsipushapis** command and then run it at step “2” on page 2053.
2. Use the **mqsipushapis** command to create or update the definition of your deployed REST APIs to an IBM API Connect server. Ensure that you have the correct parameters for the version of IBM API Connect that you want to use.
   - **Version 5**. The following example shows how you can use the command to push REST APIs to IBM API Connect Version 5 and stage the product in the catalog:

     ```
     mqsipushapis INODE --integration-server default --apic-host mysystem
      --apic-port 443 --apic-version v5 --apic-username testuser --apic-password mypassword
      --apic-org myOrg –-apic-catalog-title MyCatalog --apic-product-title MyProduct
      –-apic-product-name ProdName –-apic-product-version 2.0.0 --rest-apis myapi1:myapi2:myapi3
     ```

     You can choose to be prompted for the IBM API Connect password rather than entering it in the command, by specifying `--apic-password ""` (an empty string in double quotes).
   - **Version 5 on IBM Cloud**. The following example shows how you can use the **mqsipushapis** command to push REST APIs to IBM API Connect Version 5 on IBM Cloud and stage the product in the catalog:

     ```
     mqsipushapis --admin-host localhost --admin-port 7600
     --apic-host apimanager.eu-gb.apiconnect.cloud.ibm.com --apic-port 443
     --apic-version v5-on-ibm-cloud --apic-api-key ab209e28-6ajd-1965-8916-7457d176b0c7
     --apic-org myOrg –-apic-catalog-title MyCatalog --apic-product-title MyProduct
     ```

```
 --rest-apis myapi4:myapi5 --apic-disable-certificate-verification
```

In this example, the product name and version are not specified, so the product name is derived from the product title, and the product version defaults to `1.0.0`.

For the parameter **`--apic-host`**, enter the URL of the API Connect system. Omit the `http://` or `https://` protocol, but include the region code. In this example, the **`--apic-host`** value includes `eu-gb`, the region code for London. For more information about region codes, see https://cloud.ibm.com/docs/overview?topic=overview-whatsnew#updated-global-location-names.

- **Version 2018**. The following example shows how you can use the **`mqsipushapis`** command to push REST APIs to IBM API Connect Version 2018 and stage the product in the catalog:

```
mqsipushapis --admin-host localhost --admin-port 4415 --apic-version v2018
--apic-host myhost --apic-port 443 --apic-org v2018-org --apic-product-title myProduct2
--apic-product-name my-product-2 --rest-apis myapi1 --apic-username v2018-user --apic-
password ""
--apic-realm provider/default-idp-2 --apic-client-id clientid --apic-client-secret
 clientsecret
--apic-catalog-title Sandbox --integration-server CVS --apic-disable-certificate-
verification
```

In this example, an empty string is specified for the password parameter (`--apic-password ""`), so the system prompts for the IBM API Connect password.

The command must also include parameters for the realm, client ID, and client secret.

Client ID and Client Secret: To obtain a Client ID & Secret, you have to register a client application with the IBM API Connect environment by using the IBM API Connect command-line interface. The client application in this case is IBM App Connect Enterprise. Follow these steps to obtain a Client ID and Client Secret:

a. Create a JSON object with the following fields:

   - name
   - client_id
   - client_secret
   - client_type

   Example JSON:

```
ace-registration.json
{
  "name": "ace-registration",
  "client_id": "clientid",
  "client_secret": "clientsecret",
  "client_type": "toolkit"
```

```
}
```

b. Download and install the **CLI** version of the IBM API Connect command-line interface, as described at https://www.ibm.com/docs/en/api-connect/2018.x?topic=environment-installing-toolkit.

c. Log in to the IBM API Connect command-line interface.

```
./apic-slim login
```

d. Create an IBM API Connect registration for IBM App Connect Enterprise by running the following command in the IBM API Connect command-line interface. Specify the JSON object that you created as the input file.

```
./apic-slim registrations:create --server mycloudmanager.mydomain.com ace-
registration.json
```

If you specify an existing client_id in the JSON object, the registration process silently generates a new client_id.

e. Confirm the registration by running the following command:

```
./apic-slim registrations:get --server mycloudmanager.mydomain.com ace-reg
```

- **Version 10**. The following example shows how you can use the **mqsipushapis** command to push REST APIs to IBM API Connect Version 10 and stage the product in the catalog:

```
mqsipushapis --admin-host localhost --admin-port 4415 --apic-version v10
--apic-host myhost --apic-port 443 --apic-org v10-org --apic-product-title myProduct2
--apic-product-name my-product-2 --rest-apis myapi1 --apic-username v10-user --apic-
password ""
--apic-realm provider/default-idp-2 --apic-client-id clientid --apic-client-secret
 clientsecret
--apic-catalog-title Sandbox --integration-server CVS --apic-disable-certificate-
verification
```

In this example, an empty string is specified for the password parameter (`--apic-password ""`), so the system prompts for the IBM API Connect password.

The command must also include parameters for the realm, client ID, and client secret.

Client ID and Client Secret: To obtain a Client ID & Secret, you have to register a client application with the IBM API Connect environment by using the IBM API Connect command-line interface. The client application in this case is IBM App Connect Enterprise. Follow these steps to obtain a Client ID and Client Secret:

a. Create a JSON object with the following fields:

- name
- client_id
- client_secret
- client_type

Example JSON:

```
ace-registration.json
{
  "name": "ace-registration",
  "client_id": "clientid",
  "client_secret": "clientsecret",
  "client_type": "toolkit"
```

```
}
```

b. Download and install the **CLI** version of the IBM API Connect command-line interface, as described at https://www.ibm.com/docs/en/api-connect/10.0.x?topic=configuration-installing-toolkit.

c. Log in to the IBM API Connect command-line interface.

```
./apic-slim login
```

d. Create an IBM API Connect registration for IBM App Connect Enterprise by running the following command in the IBM API Connect command-line interface. Specify the JSON object that you created as the input file.

```
./apic-slim registrations:create --server mycloudmanager.mydomain.com ace-
registration.json
```

If you specify an existing client_id in the JSON object, the registration process silently generates a new client_id.

e. Confirm the registration by running the following command:

```
./apic-slim registrations:get --server mycloudmanager.mydomain.com ace-reg
```

For more information about creating the JSON object and client registration, see https://apic-api.apiconnect.ibmcloud.com/v10/#/documentation/authentication/auth_clientid_secret.

Note: If the IBM API Connect environment is configured with self-signed certificates or if the certificate configuration does not match the domain of the IBM API Connect environment, the command fails. Include the parameter **`--apic-disable-certificate-verification`** to instruct IBM App Connect Enterprise to ignore the errors. This parameter is not required when you connect to API Connect Version 5 on IBM Cloud.

3. When the command is run, the selected REST APIs are pushed to the IBM API Connect server:

- If there is an existing API definition on the IBM API Connect server, the existing definition is replaced by the definition of your REST API. If there is no existing API definition on the server, a new API definition is created for your REST API.
- If the specified product exists, it is updated. If the specified product does not exist, a new product is created.
- If a catalog is selected, the product is staged in the catalog.

Results

Your REST API definition is created or updated on an IBM API Connect server.

Developing integration solutions by using patterns

Create integration solutions that are used to solve a specific business problem by using patterns.

Before you begin

If you are not familiar with message flow concepts, message model concepts, and common tasks to manage message flow resources, see “Developing message flows” on page 482 and “Constructing message models” on page 2133.

About this task

A catalog of IBM App Connect Enterprise patterns, which is available on a GitHub repository, provides reusable solutions that encapsulate a tested approach to solving a common architecture, design, or deployment task in a particular context. Much of the work of design and development has been done for you when you use a pattern.

You can use these patterns unchanged, or modify them to meet your own requirements. You must create additional resources to complement the pattern and complete the solution.

Read the following sections to learn more about developing integration solutions by using patterns:

- "Patterns" on page 2057
- "Using patterns" on page 2058
- "Getting patterns from the GitHub repository" on page 2067

Patterns

A *pattern* is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context.

A pattern captures a tested solution to a commonly recurring problem, addressing the objectives that you want to achieve. The specification of a pattern describes the problem that is being addressed, why the problem is important (the value statement), and any constraints for the solution. Patterns typically emerge from common usage and the application of a particular product or technology.

An IBM App Connect Enterprise pattern can be used to generate customized solutions to a recurring problem in an efficient way. IBM App Connect Enterprise patterns are provided to encourage the adoption of preferred techniques in message flow design, to produce efficient and reliable flows. Patterns provide the following benefits:

- Give you guidance for the implementation of solutions
- Increase development efficiency, because resources are generated from a set of predefined templates
- Result in higher-quality solutions, through reuse of assets and common implementation of programming approaches, such as error handling and logging

A catalog of IBM App Connect Enterprise patterns is provided in a repository on GitHub that is accessible from the **Patterns Explorer** view in the IBM App Connect Enterprise Toolkit. The patterns are divided into pattern categories. *Pattern categories* are categories that are based on the pattern classification and structure the display in the Patterns Explorer. The catalog provides detailed help that guides you toward a suitable IBM App Connect Enterprise pattern to create resources that are used to solve a specific business problem.

You can also create your own *user-defined patterns*.

Each pattern has values that are known as pattern parameters. *Pattern parameters* are parameters that customize and configure an IBM App Connect Enterprise pattern. The pattern parameters that you configure depend on the particular pattern, and on the options that you enable for that pattern. An example of a pattern parameter is a queue name from where messages are read.

The IBM App Connect Enterprise patterns provide defaults for most pattern parameters, and help is provided to explain them. After you have configured the pattern parameters, you generate a *pattern instance project*, which contains references to all other projects in the workspace that relate to your pattern instance. A pattern instance project also contains a pattern instance configuration file that stores the pattern parameter values. This configuration file stores the pattern parameters that you configured. Generating a pattern instance project also creates one or more additional IBM App Connect Enterprise projects that typically contain message flows and other IBM App Connect Enterprise resources that implement the pattern.

You can open the pattern instance configuration file at any time to see the values of the pattern parameters. When a pattern configuration file has been reopened, you can regenerate the IBM App Connect Enterprise projects. Regeneration deletes the generated IBM App Connect Enterprise projects and re-creates them from scratch. The pattern instance project contains an HTML summary file and a pattern instance configuration file. The summary file has a section that explains additional tasks that might be required, such as creating queues.

You can create resources from each IBM App Connect Enterprise pattern more than once to give you unique pattern instances, each with a different configuration. The pattern parameters that you can configure depend on the particular pattern, and also the options that you enable for that pattern.

Some of the pattern parameter settings can affect the resources that are generated. For example, if you enable logging and error handling, the generated projects contain additional message flows and ESQL scripts.

In all cases, the specification of both the problem and the solution are indispensable parts of the pattern definition.

Using patterns

Use patterns that are supplied with the IBM App Connect Enterprise Toolkit to create resources that are used to solve a specific business problem.

Each IBM App Connect Enterprise *pattern* is designed to address a specific business problem. To find out about the IBM App Connect Enterprise patterns that are supplied on GitHub, see "Getting patterns from the GitHub repository" on page 2067.

Each pattern has values known as *pattern parameters* that you use to create the pattern resources for use in your environment. The pattern parameters that you can configure depend on the specific pattern, and also the options that you enable for that pattern; for example, logging.

When you select a pattern from the Patterns Explorer, the **Pattern Specification** tab gives details about the purpose and configuration of each pattern parameter associated with a pattern, the effects and consequences of changing them, and any postrequisite tasks. All patterns have pattern parameter properties that distinguish one application of a pattern template from another.

All patterns are either abstract or implementations and as you move down the tree the patterns become more specific. At any level in a tree a pattern can be an implementation.

You can apply pattern implementations only. Pattern implementations map to a complete specification with prerequisite and postrequisite tasks, and pattern parameter details. Pattern implementations have a **Create New Instance** button in the **Pattern Specification** tab. *Abstract* patterns, which cannot be applied, do not have a **Create New Instance** button in the **Pattern Specification** tab.

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

Choosing a pattern

Select a pattern in the **Patterns Explorer** view to create resources to solve a specific business problem.

Before you begin

Before you complete this task, read the following overview topics about patterns:

- "Patterns" on page 2057
- "Using patterns" on page 2058

About this task

To create resources by using a pattern, complete the following steps:

Procedure

1. Open the **Patterns Explorer** view by using one of the following methods:
   - In the Application Development view, click **New pattern instance** in the **Pattern Instances** section.
   - Click the **Patterns Explorer** tab.
   - Click **Quick Starts**, then click **Start from patterns**.
   - Click the **Open Patterns Explorer** icon in the task bar ().

   The **Patterns Explorer** view is displayed.
2. In the **Patterns Explorer** view, click **Patterns**.

   The **Pattern Specification** tab displays information about the patterns. For example, to find out about message-based integration patterns, complete the following steps:

   a) Click **Message-based Integration**.

   The **Pattern Specification** tab displays information about the selected pattern, and because it is an abstract pattern the **Create New Instance** button is not displayed.

   b) Under **Message-based Integration** click another pattern, for example **Message Correlator**.

   The **Pattern Specification** tab displays information about the selected pattern, and because it is an abstract pattern the **Create New Instance** button is not displayed.

   c) Under **Message Correlator** click another pattern, for example **MQ request-response with persistence**.

   The **Pattern Specification** tab displays information about the selected pattern, and because it is a pattern implementation the **Create New Instance** button is displayed.
3. Before you use your pattern, read "Tasks to complete before applying the pattern" and "Constraints on the use of the pattern".
4. To use a pattern implementation from the **Pattern Specification** tab, click **Create New Instance**.

   The **Create New Instance** window opens.
5. Enter a new pattern instance name.

   You cannot use the following names:
   - The name of an existing project in the current workspace.
   - On Windows only: MS-DOS device driver names. For more information, see the article on the Microsoft support site: MS-DOS device driver names cannot be used as file names.
6. Click **OK**. If you click **Cancel**, no changes are made to the workspace.

   The **Configure Pattern Parameters** page is displayed.
7. On the **Configure Pattern Parameters** page, complete all required pattern parameter fields.

   Configure pattern parameters by entering the appropriate values. If you do not want to generate the pattern immediately, save the parameters to edit later. Before you apply a pattern, enter a value for each pattern parameter. You can accept a default value, if one is offered. Empty strings are permitted for some fields.

   You can reuse a saved configuration by editing the parameters, see “Editing and regenerating a pattern” on page 2065.

   Parameter groups might be marked with error markers. You can click the error marker to move to the first field that has an error.

What to do next

Now complete the following task:

- “Generating a pattern instance” on page 2064

Working with patterns in the Application Development view

Using the Application Development view to create patterns.

Before you begin

Before completing this task, read the following overview topics about patterns:

- "Patterns" on page 2057
- "Using patterns" on page 2058

About this task

Pattern instances are shown under the Pattern Instances folder in the Application Development view. If you have not created any pattern instances yet, the folder is not visible. Similarly, pattern authoring projects are shown under the Pattern Authoring folder.

The Patterns Explorer is a separate tab. When you select a pattern from the Patterns Explorer, the specification for that pattern is shown in the editor view. When you click **Create New Instance**, a pattern configuration opens in the editor. You configure your parameters for the pattern, then click **Generate**. A pattern instance is created, and is shown under the Pattern Instances folder in the Application Development view. A working set is also created for the pattern and is immediately active, so that you can see only the resources that are associated with your pattern. To see all resources in your workspace, click the **Remove all filters** icon  on the toolbar of the Application Development view.

What to do next

Now complete the following task:

- "Choosing a pattern" on page 2058

Adding or removing project references in a pattern instance project

A pattern instance project contains references to regular projects and they are listed in the Project References category of a pattern instance project.

About this task

The Project References category under a pattern instance project in the Application Development view shows the same project references as the Project References page in the Properties window.

To add another project to the list of project references for your pattern instance project, complete the following steps:

Procedure

1. Expand the Pattern Instances folder in the Application Development view.
2. Open the pattern instance project and select the **Project References**.
3. Right-click the **Project References** and then select **Properties**.
4. Add projects by selecting them in the **Project References**.
   You can remove projects in the same way by deselecting them.

Going to a referenced project

How to display your pattern instance project in the Application Development view.

About this task

To display your pattern instance project complete the following steps.

Procedure

1. Expand the Pattern Instances folder in the Application Development view.
2. Expand your pattern instance project, and expand **Project References**.
3. Right-click your pattern instance project reference.
4. Select **Go to Referenced Project**.
   The referenced flows project is selected in the Application Development view.

Showing a working set for a pattern instance project

When a pattern instance project is created by the Patterns Explorer, a working set of the same name is generated, and all projects that are part of that pattern instance are placed inside it.

About this task

To view all of the projects in your pattern instance:

Procedure

1. Click the arrow on the toolbar of the Application Development view, then click **Select Working Set**.
   All working sets are displayed, including those for pattern instances.
2. Select the relevant working sets and click **OK**.
   Only the resources associated with the selected working sets are shown in the Application Development view.
3. To see all resources in the workspace, click the arrow on the Application Development view toolbar, then click **Deselect Working Set**.
   Alternatively, click the **Remove all filters** icon  on the toolbar of the Application Development view.

Focusing on a pattern instance

Several ways are available for you to focus on a particular pattern instance.

Procedure

To focus on your pattern you can use one of the following methods:

- In the Pattern Instances section, right-click your pattern instance project, select **Focus on Pattern Instance**.
- In the toolbar of the Application Development view, click the down arrow and select your pattern instance name from the menu.

The navigator refreshes to show the currently selected working set and its associated projects.

To confirm that your pattern instance project is currently in focus, right-click your pattern instance project, ensure that the check box to the left of the **Focus on Pattern Instance** menu item is selected. If you clear this check box, the navigator shows all of the projects.

If the navigator has focus on your working set, if you click the down arrow in the IBM App Connect Enterprise Development toolbar, your working set menu item is selected in the menu.

Creating a working set and focus on a pattern instance

Creating a working set for an imported project interchange file that contains a pattern instance project and its associated projects.

About this task

If you import a project interchange file that contains a pattern instance project and its associated projects, a working set is not automatically created for the pattern instance project. Therefore, you must create a working set.

Procedure

In the Pattern Instances section of the Application Development view, right-click the pattern instance project, and select **Create Working Set and Focus on Pattern Instance** in the menu.

When this item is selected, a working set is created with the same name as the pattern instance project, the pattern instance project and its associated projects are added to that working set, and the working set is given focus in the navigator.

Deleting a pattern instance project

Pattern instance projects typically contain references to regular projects. Therefore, when you select a pattern instance project for deletion you must decide whether to delete all of the referenced projects from the file system. Some pattern instance projects have no project references.

About this task

Use the following instructions depending on the type of pattern instance that you want to delete:

- "Deleting a pattern instance project with project references" on page 2062
- "Deleting a pattern instance project with no project references" on page 2063

Deleting a pattern instance project with project references

Pattern instance projects typically contain references to regular projects. Therefore, when you select a pattern instance project for deletion you must decide whether to delete all of the referenced projects from the file system.

About this task

Pattern authors can configure projects in a pattern authoring project so that they are not overwritten when a pattern instance is regenerated. Projects that have been configured in this way are not deleted when a pattern instance is deleted.

Use the following steps to delete a pattern instance project that has project references:

Procedure

1. In the Application Development view, open the Pattern Instances section.
2. Right-click the pattern instance project, or projects, that you want to delete, click **Delete**.

   The **Confirm Project Delete** window opens, displaying any projects that are referenced by the pattern instance project. The **Also delete the projects referenced by this pattern instance project** check box is selected and the **Also delete contents in the file system** check box is cleared by default.
3. Choose one of the following options:
   - To delete the pattern instance project, or projects, and the project references, but leave the contents in the file system, keep the default settings, click **Yes**. The pattern instance project, or projects, and the project references still exist in the file system.
   - To delete the pattern instance project, or projects, and the project references from the workspace and the file system, ensure that **Also delete the projects referenced by this pattern instance project** and **Also delete contents in the file system** are both selected, click **Yes**. The working set is deleted.
   - To delete your pattern instance project, or projects, from the workspace but not from the file system and to keep the project references in the workspace and file system, ensure that **Also delete the projects referenced by this pattern instance project** and **Also delete contents in the file system** are both cleared. The working set is deleted.
   - If you decide not to delete the pattern instance project, or projects, click **No**.

Deleting a pattern instance project with no project references
Decide whether to delete your pattern instance project, which has no project references, from both the workspace and the file system.

About this task

Use the following steps to delete a pattern instance project that has no project references:

Procedure

1. In the Application Development view, open the Pattern Instances section.
2. Right-click the pattern instance project, or projects, that you want to delete, click **Delete**. The **Confirm Project Delete** window opens, displaying the pattern instance project. The **Also delete contents under *pattern _instance_name*** check box is cleared by default. The default is to delete the pattern instance project from the workspace, but not to delete it from the file system.
3. Choose one of the following options:
   - To delete the pattern instance project, or projects, from the workspace, but not from the file system, click **Yes**. The working set is also deleted.
   - To delete the pattern instance project, or projects, from the workspace and from the file system, select the **Also delete contents under *pattern_instance_name*** check box, click **Yes**. The working set is also deleted.
   - If you decide not to delete the pattern instance project, or projects, click **No**.

Deleting projects

If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

About this task

Use the following instructions depending on whether the project that you want to delete is referenced by a pattern instance project:

- "Deleting projects that are referenced by a pattern instance project" on page 2063
- "Deleting projects that are not referenced by a pattern instance project" on page 2063

Deleting projects that are referenced by a pattern instance project
If you want to delete an application, library, or integration project, you must take into account whether it is referenced by a pattern instance project.

About this task

If you delete an application, library, or integration project that is referenced by a pattern instance project, the project reference is removed from the pattern instance project. Therefore, before you delete an application, library, or integration project, ensure that it is not referenced by another project.

Deleting projects that are not referenced by a pattern instance project
If you want to delete a project, you must take into account whether it is referenced by a pattern instance project.

About this task

Complete the following steps to delete your project if it is not referenced by a pattern instance project:

Procedure

1. In the Application Development view, select the project that you want to delete, then click **Delete**.

   The **Confirm Project Delete** window opens asking whether you are sure that you want to delete the project. The default action is to delete the project from the workspace, but not to delete it from the file system. Therefore, by default, the **Also delete content under *project_name*** check box is cleared.
2. Choose one of the following options:
   - To delete the projects from the workspace, but not to delete it from the file system, ensure that the **Also delete content under *project_name*** check box is cleared.
   - To delete the projects from the workspace and delete it from the file system, ensure that the **Also delete content under *project_name*** check box is selected.
   - If you decide not to delete the projects, click **No**.

Generating a pattern instance

Generate resources from the pattern.

Before you begin

You must have chosen pattern. For more details, see "Choosing a pattern" on page 2058.

About this task

The **Generate** button is unavailable until all required parameters are complete.

To generate the IBM App Connect Enterprise Toolkit resources for the pattern, complete the following steps.

Procedure

On the **Configuration** page, click **Generate**.

Focus is automatically moved to the Application Development view, which lists all the new resources that have been generated.

All of the other resources are filtered out in this view.

Results

Following a successful **Generate** action:

- A pattern instance project is created in your workspace and is displayed in the Application Development view. The pattern instance project contains a pattern instance configuration file and a summary file. The summary file describes the tasks that have been generated, and the tasks that are still required.
- All the resources generated for a pattern instance are packaged into their own working set and the Application Development view displays only the resources in this working set. The name of the working set is always the same as the pattern instance name provided. This name must always be unique within the workspace.
- You can regenerate the pattern by using the same instance name, but, if you confirm the regeneration, existing projects are deleted, see "Editing and regenerating a pattern" on page 2065.

What to do next

Now complete the following task:

- "Reviewing the pattern instance summary and tasks" on page 2065

Editing and regenerating a pattern

You can regenerate a pattern instance after you have edited a saved configuration in a pattern instance project.

About this task

Before you begin:

You must be in the IBM App Connect Enterprise Toolkit and have an existing pattern configuration file in a pattern instance project.

Procedure

1. Open an existing pattern configuration by double-clicking the configuration XML file in a pattern instance project.

   The **Configure Pattern Parameters** page of the editor displays parameter values for that configuration instance. The Integration node navigation pane is in focus. The title of the **Configure Pattern Parameters** page for the parameter reflects the pattern instance name (not the pattern name).
2. View and edit the pattern parameters. You can now choose either of the following options:
   - Close the editor without saving the changes.
   - Select **Generate**. Resources, such as projects, already exist in the workspace. You are warned that these resources will be deleted and you can choose whether to continue. If you continue, the resources are regenerated; the existing resources are deleted and re-created. If you decide not to continue, the pattern configuration remains open and no changes are made to the pattern resources.

     When you make changes in the Pattern Configuration editor, the summary file and configuration XML file might be out of date because the *instance_name*`_configuration.xml` and the *instance_name*`_summary.html` files do not match the generated resources. The **Problems** view displays any warning messages. Double-click the warning message to open the relevant editor for the resource.

What to do next

Now complete the following task:

- “Reviewing the pattern instance summary and tasks” on page 2065

Reviewing the pattern instance summary and tasks

Showing the summary page for your pattern instance project.

Before you begin

You must have completed the following tasks:

- “Generating a pattern instance” on page 2064

About this task

After generating the pattern instance, you might have to do additional tasks to complete your solution and to support its operation following instance. These tasks are located in a summary file that is created in the pattern reference project, which has the same name as the instance name.

The summary file lists the resources that were created for the pattern instance, based on the pattern parameter values that you entered in the **Configure Pattern Parameters** page. The summary also lists any outstanding tasks that you must complete to make the solution operational. For example, you might have to configure the message set details that are required by one or more input nodes that form part of the overall solution, or create IBM MQ queues for the nodes.

When a pattern has been configured and generated once, it contains both a configuration file and a summary file. If you edit and save the configuration file, both the configuration and the summary files display a warning symbol and a warning also shows in the **Problems** window indicating that the two files are out of sync. To get the files back in sync, regenerate the pattern.

To view the summary and tasks:

Procedure

1. Click **Windows** > **Show view** > **Tasks** to open the **Task** view.
2. In the **Path** column, find your pattern instance project name, which includes the pattern instance name that you used when you created the pattern instance.
3. To open the Summary page, double-click anywhere on the task, or right-click and select **Go To**.
4. Complete the tasks that are shown in the summary file.

What to do next

Now complete the following task:

- "Adding resources to a BAR file" on page 2470

Importing an existing configuration

You can import a pattern configuration XML file from the workspace or file system and populate your current pattern instance with its values.

Before you begin

Before you start:

Before completing this task, you must have a compatible pattern configuration.

You must be in the IBM App Connect Enterprise Toolkit and have a pattern instance open.

Procedure

1. Open your current configuration. On the **Configure Pattern Parameters** page, click the **Open existing configuration** icon.

   The **Open Pattern Configuration** window opens.
2. You can select an existing pattern XML configuration file either from a workspace or file system location.

   Only configuration files created by the same pattern can be imported.

   - To select from a list of compatible XML configuration files that are already in the workspace, click **Workspace**. If you select a valid XML configuration file, the editor is configured with the values from the existing configuration file. You can then choose one of the following options:
     - Save the changes by clicking **OK**.
     - Return to the **Configure Pattern Parameters** page and discard the changes by clicking **Cancel**.
   - To select an XML configuration file from the file system by using a standard file system window, click **File System**. Select the file that you want to use, click **Open**.

   The configuration XML is validated against the corresponding pattern schema, if you select a configuration that is not valid, for example, from a different pattern, you receive notification. You can then either select the correct file, or cancel the action.
3. Share the pattern instance projects.

   To share pattern instance projects between IBM App Connect Enterprise Toolkit workspaces, you can export the pattern instance project into a directory in the file system or into a Project Interchange File (PIF). You can then import the pattern instance project into another workspace.

What to do next

Now complete the following task:

- "Generating a pattern instance" on page 2064

Getting patterns from the GitHub repository

You can download and install existing patterns from the OT4I GitHub Pattern Repository by using the IBM App Connect Enterprise Toolkit.

About this task

From the **Patterns Explorer** in the IBM App Connect Enterprise Toolkit, complete the following steps to download and install a pattern:

Procedure

1. Right-click **OT4I GitHub Pattern Repository** under **Pattern Repositories**, and click **Connect** to display the categories of the patterns that are available in the repository.
2. Expand a pattern category to list the patterns that are available in that category.

   Each pattern name in the list is appended with **[Not Installed]** if the pattern is not installed in the IBM App Connect Enterprise Toolkit, or **[Installed]** if the pattern is already installed.
3. Right-click the pattern that you want to download and click **Download and Install**.

   This menu option is available only if the pattern is not already installed.
4. When the security warning is displayed, click **OK**.

   The pattern is downloaded and installed, the pattern name in the list is appended with **[Installed]**, and the pattern is listed under **Patterns** in the **Patterns Explorer**.
5. Optional: If you want to uninstall and remove a pattern from the IBM App Connect Enterprise Toolkit, right-click the pattern name in **Pattern Repositories** > **OT4I GitHub Pattern Repository** and click **Uninstall**.

   The pattern is uninstalled and removed from the list of patterns under **Patterns**, and the pattern name in the repository is appended with **[Not Installed]**.

   Note: Uninstalling a pattern from the IBM App Connect Enterprise Toolkit does not affect any instances that are created from the pattern.

What to do next

Read the pattern documentation and create a pattern instance to explore the pattern functionality; see "Using patterns" on page 2058

Note: From the **Patterns Explorer** you also have access to the **Experimental OT4I GitHub Pattern Repository**. The **Experimental OT4I GitHub Pattern Repository** contains patterns that are available for testing but are not fully-supported.

User-defined patterns

A *user-defined pattern* extends the function of IBM App Connect Enterprise so that you are able to create patterns that you can reuse within your organization.

The *pattern author* creates a user-defined pattern to meet a business or technical requirement, and the *pattern user* configures a user-defined pattern that the pattern author has created. The user-defined pattern is available to a pattern user in the Patterns Explorer view in the Integration Development perspective of the IBM App Connect Enterprise Toolkit.

To create user-defined patterns, you must provide the implementation of the solution that reflects good practice in your organization. This implementation is known as an *exemplar*. An exemplar is a project that holds content for a pattern. An exemplar contains message flows and other resources, such as source

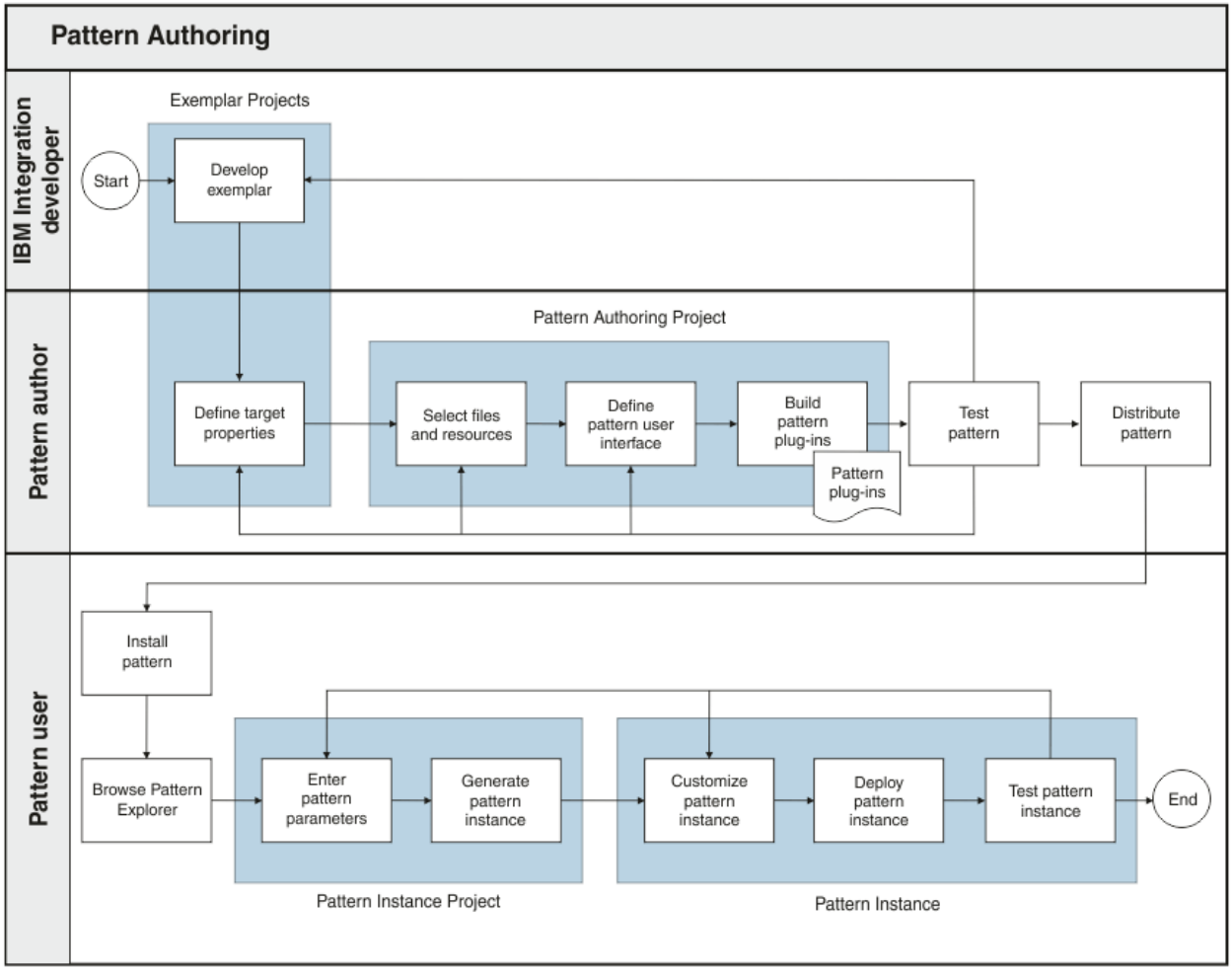
code, Java classes for JavaCompute nodes, ESQL modules, message maps, test client, XML files, and style sheet files. Exemplars are used to create pattern plug-ins by configuring a *pattern authoring project*.

You can use the Pattern Authoring editor to configure a pattern and create the pattern plug-ins that implement the pattern. You can distribute the pattern plug-ins so that other people can use your pattern. Your pattern is displayed in the **Patterns Explorer** view and can be used in the same way as the built-in patterns.

To create a user-defined pattern, see "Creating a user-defined pattern" on page 2068.

Creating a user-defined pattern

The workflow showing the actions required for pattern authoring.



The three stages of creating a user-defined pattern are performed by the IBM App Connect Enterprise developer, the pattern author, and the pattern user.

1. The IBM App Connect Enterprise developer develops the exemplar. The exemplar is fundamental to the pattern authoring process. It assumes that the exemplar is the starting point for a pattern. Some modifications might be required to the exemplar to prepare it for pattern authoring.
2. The pattern author creates a pattern plug-in from the exemplar. Authoring a pattern is a design activity. Some of the pattern authoring focuses on the resources in IBM App Connect Enterprise, for example, defining the target properties for the pattern and configuring the user interface that is presented to the pattern user. The pattern author ensures that the pattern can be customized by pattern users, if required, and that these customizations are not overwritten if the pattern user regenerates an instance of the pattern. One possible approach to customization is to include in the message flows of the pattern one or more subflows that are intended to be customized by pattern users. The pattern author

includes these subflows in a separate project within the pattern and configures the project so that it is not overwritten when a pattern instance is regenerated.

The pattern author then shares the user-defined pattern with the pattern user. Patterns are most useful when they are shared with a user base. This user base can be within an organization, or a broader community, for example, open source licensing.

3. The pattern user receives a user-defined pattern, customizes it, and uses it in accordance with the requirements of the organization.

Pattern authoring

To create a user-defined pattern, complete the following tasks. "Extending a user-defined pattern" is an optional set of tasks that you can use to add features to your user-defined pattern. For example, by extending a user-defined pattern you can create user-defined patterns that include parameters that pattern users can modify.

1. “Creating a pattern authoring project” on page 2069
2. “Selecting the source files to use for a user-defined pattern” on page 2070
3. Optional: “Adding and removing the project references for a user-defined pattern” on page 2070
4. Optional: “Configuring a project used in a user-defined pattern” on page 2071
5. Optional: “Extending a user-defined pattern” on page 2072
6. “Building pattern plug-ins” on page 2126
7. “Testing a user-defined pattern” on page 2126
8. “Packaging and distributing pattern plug-ins” on page 2127

When you have created pattern plug-ins for your user-defined pattern, pattern users can install them to use in IBM App Connect Enterprise projects, see “Downloading and installing a pattern archive” on page 2130.

Creating a pattern authoring project

Create a pattern authoring project and choose an exemplar for the project.

About this task

Procedure

To create a pattern authoring project:

1. Click **File** > **New** > **Other**.

   A window opens in which you can select a wizard.
2. Expand **Integration Bus - Application Development**, select **Pattern Authoring Project**, then click **Next**.

   The **New Pattern Authoring Project** wizard opens.
3. Enter a pattern name and a project name for your pattern, then click **Next**.
4. Select the project references that you require.

   These references are the exemplars for the user-defined pattern.
5. Click **Finish**.

Results

Your pattern is displayed under the Pattern Authoring folder in the Application Development view, and is opened in the Pattern Authoring editor.

What to do next

Select the source files. For more information, see "Selecting the source files to use for a user-defined pattern" on page 2070.

Selecting the source files to use for a user-defined pattern

Select the source files to include in your pattern.

Before you begin

Complete the following task:

- "Creating a pattern authoring project" on page 2069

Procedure

To select the source files to use in the user-defined pattern:

1. In the Pattern Authoring editor, click the **Source Files** tab.
2. In the **Select Source Files** section, select the source files in the referenced projects that you want to include in your pattern.

What to do next

- Extend the pattern, see "Extending a user-defined pattern" on page 2072.
- Change the project references, see "Adding and removing the project references for a user-defined pattern" on page 2070.
- Build the pattern, see "Building pattern plug-ins" on page 2126.

Adding and removing the project references for a user-defined pattern

Change the project references for a user-defined pattern so that you can change the source files in the user-defined pattern.

Before you begin

Complete the following task:

- "Creating a pattern authoring project" on page 2069

Procedure

To add or remove project references:

1. In the Pattern Authoring editor, click the **Source Files** tab. Click **Change Project References**.
2. To add a new project reference, select the project reference that you want to add and click **OK**.

   If you add a new project reference, none of the source files within the project reference are selected. To select source files for the new project reference, see "Selecting the source files to use for a user-defined pattern" on page 2070. If your user-defined pattern requires referenced projects but they are not included as source files in the pattern, you must distribute the referenced projects to pattern users to ensure the user-defined pattern works.
3. To remove a project reference, clear the project reference that you want to remove and click **OK**.

   If you remove a referenced project from the workspace, the source files are removed from the pattern authoring project but the project reference remains. If you later add the project back into the workspace, the source files are added to the pattern authoring project.

What to do next

- You can extend the pattern; see "Extending a user-defined pattern" on page 2072.
- If the pattern is complete, you can now build the pattern; see "Building pattern plug-ins" on page 2126.

Configuring a project used in a user-defined pattern

Configure a project that is included in a user-defined pattern to control whether the project is overwritten when a pattern user regenerates a pattern, and to set the name of the project.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Adding and removing the project references for a user-defined pattern" on page 2070

About this task

When a pattern user generates a user-defined pattern, copies of projects that are selected as source files in the pattern authoring project are created in the pattern instance. By default, if a pattern user customizes the projects in an instance of a user-defined pattern and then regenerates the pattern instance, those customizations are overwritten. As a pattern author, you can configure projects in a user-defined pattern project so that they are not regenerated when a pattern user regenerates a pattern instance. Pattern users can then customize these projects and regenerate pattern instances without losing the customizations. Projects that are configured so they are not overwritten are also not overwritten if a new version of the pattern is installed before the pattern user regenerates a pattern instance.

As a pattern author, you can specify how a project that is contained in a user-defined pattern is named when a pattern instance is generated. By default, the pattern instance name is added as a prefix to the project name. You can turn off this prefix. If you generate more than one instance of a user-defined pattern that contains a project name without a prefix, all instances of the pattern use the same copy of the project.

As a pattern author, you can name a project in a user-defined pattern by using the value of a pattern parameter. The project name is determined after all transformation expressions are calculated.

Procedure

To configure a project used in a user-defined pattern:

1. In the Pattern Authoring editor, in the **Select Source Files** section, select the project you want to configure and click **Edit**, or double-click the project you want to configure.

   The **Edit Project** window opens.
2. To configure a project so that when a pattern user regenerates a pattern instance it is not overwritten, select **Pattern users will modify this project**.

   If the project does not exist, it is generated when a pattern user generates an instance of the pattern. If the project exists it is not overwritten. If the project exists, but files within it have been deleted, these files are created.
3. To configure a project so that when a pattern user generates a pattern instance the project name does not start with the name of the pattern instance, clear **Prefix the project name with the pattern instance name**.
4. To configure a project so that when a pattern user generates a pattern instance the project name is set to the value of a pattern parameter, click **Select**. In the **Select Pattern Parameter** window, select the parameter you want to use to set the project name, then click **OK**.
5. Click **OK**.

What to do next

- Extend the pattern, see "Extending a user-defined pattern" on page 2072
- Build the pattern, see "Building pattern plug-ins" on page 2126

Extending a user-defined pattern

Extend a user-defined pattern to customize its behavior so that it is easier to use or to provide guidance to pattern users.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070

About this task

A basic user-defined pattern is a copy of an existing message flow, but by extending a user-defined pattern you can allow the pattern user to customize the pattern.

Procedure

You can extend a user-defined pattern in the following ways:

- Create documentation for the pattern.

  Creating documentation provides guidance to your pattern users. To update the pattern documentation, see "Creating documentation for a pattern" on page 2073.
- Define the target properties.

  A user-defined pattern can change the value of user-defined properties, promoted node properties, and node properties in a message flow. A property that is changed by a pattern is called a target property. Pattern users create a pattern instance by configuring pattern parameters. The values of the pattern parameters that are configured by a pattern user can be used to configure the target properties in the pattern instance. To define target properties, see "Defining the target properties" on page 2074.
- Define the user interface.

  You can customize how the pattern parameters are displayed to pattern users by defining the user interface. For example, you can rename or group pattern parameters, and you can define default values for pattern parameters. To define the user interface, see "Defining the user interface" on page 2074, "Adding and editing parameter groups" on page 2076, and "Enabling parameter groups" on page 2077.
- Configure the categories in the Patterns Explorer view.

  When a pattern user imports a user-defined pattern, it is shown in the Patterns Explorer view. You can select which category the user-defined pattern is assigned to in the Patterns Explorer view and create new categories. To configure the categories, see "Creating and configuring categories" on page 2079.
- Configure the SOAP nodes.

  To use SOAP nodes in user-defined patterns, see "Configuring SOAP nodes for user-defined patterns" on page 2080.
- Change pattern parameter IDs.

  Pattern parameter IDs are used to refer to pattern parameters in XPath, Java, and PHP code. You can write the code to modify pattern parameters or pattern instances. Default parameter IDs are assigned, but you can change them to custom IDs. To change parameter IDs, see "Changing pattern parameter IDs" on page 2081.
- Control the creation of projects in the pattern.

  You can use XPath expressions to control whether a project in a user-defined pattern is created based on the values of pattern parameters. To set up project creation expressions and then test the expressions, see "Creating a project used in a user-defined pattern" on page 2082 and "Testing a project creation expression" on page 2083.

- Transform the pattern parameters.

  You can calculate values for pattern parameters from other pattern parameters by using XPath expressions. To set up transformations for pattern parameters and then test the transformations, see “Transforming pattern parameters” on page 2084 and “Testing a transformation expression” on page 2085.
- Enable or disable the pattern parameters.

  You can use XPath expressions to control whether a pattern parameter can be modified by a pattern user based on the values of other pattern parameters. To set up enabling expressions for pattern parameters and then test the expressions, see “Enabling pattern parameters” on page 2086 and “Testing an enabling expression” on page 2087.
- Use enumerated values for the pattern parameters.

  You can set up enumerated types for pattern parameters so that pattern users have a predefined list of values for the pattern parameters. If a target property has a list of possible values, the pattern authoring tool generates an enumerated type for that target property. You can modify the enumerated type that is automatically created. To use enumerated types see “Using enumerated values for pattern parameters” on page 2088 and “Creating enumerated types for pattern parameters” on page 2088.
- Use tables for the pattern parameters.

  You can set up table types for pattern parameters by defining the columns for the table. Pattern users can then enter one or more rows of data in the table in your user-defined pattern. To use table types see “Using tables for pattern parameters” on page 2091 and “Adding pattern parameter table types” on page 2091.
- Use user-defined editors for the pattern parameters.

  You can create user-defined editors for pattern parameters in your user-defined patterns. Pattern users use these editors when they enter values for pattern parameters in an instance of your user-defined pattern. To use user-defined editors, see “Using user-defined editors for pattern parameters” on page 2093.
- Modify pattern instances by using Java or PHP.

  You can write code in Java or PHP that modifies pattern instances when a pattern user generates an instance of a user-defined pattern. For example, you can write code to modify the structure of a message flow based on the values of pattern parameters. To use Java or PHP in user-defined patterns, see “Modifying pattern instances by using Java or PHP” on page 2105.

What to do next

After extending your user-defined pattern, you must build the pattern plug-in, see “Building pattern plug-ins” on page 2126.

Creating documentation for a pattern

When you create a pattern authoring project, a default HTML pattern specification is created as part of the project. You can edit this pattern specification to document your user-defined pattern.

Before you begin

Complete the following task:

- “Creating a pattern authoring project” on page 2069

About this task

Procedure

To edit the pattern specification, complete the following steps.

1. Expand the Pattern Authoring folder in the Application Development view.
2. Expand your pattern authoring project, and expand **Pattern Specification**.

3. Right-click `overview.htm` and select **Open With** > **HTML Editor**.

   The default user-defined pattern specification is shown in the HTML editor.
4. Edit the content of the pattern specification in the HTML editor.

   The default specification contains example links to suggested HTML files. To ensure that these links work correctly, you must create the required files and edit the links in the default specification. If you do not require these additional pages, delete the example links from the default specification.
5. To save the changes, click **File** > **Save**.
6. To view the updated pattern specification, click the **Categories** tab of the Pattern Authoring editor.

Defining the target properties

Identify the target properties in the workspace project so that you can define the pattern interface and pattern users can modify the pattern parameters.

Before you begin

Create the pattern authoring project and select the source files. For information about creating a pattern authoring project, see "Creating a pattern authoring project" on page 2069. For information about selecting the source files, see "Selecting the source files to use for a user-defined pattern" on page 2070.

Procedure

To identify the target properties that you want to use in the workspace project:

1. In the Application Development view, double-click the message flow that you want to use.
2. Open the **Select Target Properties** window by completing one of the following steps:
   - To select a node property as a target property, in the Message Flow editor, right-click the node that you want to use and click **Pattern** > **Select Target Properties**.
   - To select a user-defined property as a target property, in the Message Flow editor, right-click the canvas and click **Pattern** > **Select Target Properties**.
3. In the **Select Target Properties** window, select the target properties that you want to use in the pattern.
4. Close the **Select Target Properties** window.
5. Save the message flow.

Results

The selected target properties are displayed in the Target Properties panel of the **Source Files** tab in the Pattern Authoring editor.

What to do next

Define the user interface; see "Defining the user interface" on page 2074.

Defining the user interface

To control how pattern users view and edit pattern parameters in the Pattern Instance editor after the pattern user has created an instance of a user-defined pattern, you must define the user interface.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

When a pattern user creates an instance of a user-defined pattern, the parameter groups and pattern parameters that are defined in the **Pattern Configuration** tab of the Pattern Authoring editor are displayed in the Pattern Instance editor. If the **Pattern Configuration** tab displays groups or parameters that you do not want in the final pattern, you must delete them from the **Pattern Configuration** tab before building the pattern plug-ins.

To define the user interface, open the **Pattern Configuration** tab of the Pattern Authoring editor. In the **Groups and Parameters** list, entries are shown in a hierarchy in which groups contain pattern parameters, and pattern parameters contain target properties.

Procedure

To define the user interface, choose one or more of the following options:

1. You can define the groups in which pattern parameters are displayed and assign target properties to pattern parameters:
   a) To define the groups in which pattern parameters are displayed in the Pattern Instance editor, drag pattern parameters to groups.

      To add new groups or edit existing groups, see "Adding and editing parameter groups" on page 2076. To control whether parameter groups are enabled by using XPath expressions, see "Enabling parameter groups" on page 2077.

   b) To assign target properties to pattern parameters, drag target properties to pattern parameters.

      If the target property and pattern parameter do not use a compatible editor, an error message is displayed; see step "5" on page 2076 to change the parameter editor. If you drag multiple target properties to one pattern parameter, all target properties in that pattern parameter are populated with the same value that is entered for the pattern parameter by the pattern user.
2. To add a new pattern parameter, click **Add Parameter**, and click **OK**.

   To configure the new parameter, see steps "4" on page 2075 and "5" on page 2076.
3. To remove a pattern parameter or parameter group, select the pattern parameter or parameter group to remove and click **Delete**.
   - To remove a parameter group it must have no pattern parameters assigned to it.
   - To remove a pattern parameter it must have no target properties assigned to it. To remove a target property so that you can remove a pattern parameter, clear the target property in the Message Flow editor of the Application Development view. For more information about defining target properties, see "Defining the target properties" on page 2074. When you remove a target property in the Message Flow editor, the associated pattern parameter and parameter group are not automatically deleted in the **Pattern Configuration** tab of the Pattern Authoring editor.
4. You can change the name of a pattern parameter, create help text, create a field prompt, and configure parameter options in the **Basic** tab. In the **Groups and Parameters** list, select the pattern parameter and click **Edit**, or double-click the pattern parameter. The **Edit Parameter** window opens.
   - To change the name of the pattern parameter, enter a name for the pattern parameter in the **Display Name** field and click **OK**. The name for the pattern parameter is changed to the new name.
   - To change parameter options, select or clear the **Hide the parameter** and **Mandatory parameter** check boxes, as required.
   - To enter a field prompt, type a text into the **Field prompt** field.
   - To provide help text for the pattern parameter, type the text in the **Help Text (HTML)** field.
   - To close the **Edit Parameter** window after you have completed your changes, click **OK**.

5. You can change the parameter editor, and set a default value for a pattern parameter in the **Editor** tab. In the **Groups and Parameters** list, select the pattern parameter and click **Edit**, or double-click the pattern parameter. The **Edit Parameter** window opens. Click the **Editor** tab.
   - To change the parameter editor, select the required editor in the **Parameter editor** list.
     - To use an enumerated type for the parameter, select **Drop Down Selection** in the **Parameter editor** list, then select an enumerated type for the parameter in the **Type selection** field. For more information, see "Using enumerated values for pattern parameters" on page 2088.
     - To use a table type for the parameter, select **Table Editor** in the **Parameter editor** list, then select a table type for the parameter in the **Type selection** field. For more information, see "Using tables for pattern parameters" on page 2091.
     - To use a user-defined editor for the parameter, select **User-Defined Editor** in the **Parameter editor** list. For more information, see "Using user-defined editors for pattern parameters" on page 2093.
   - To enter a default value for a pattern parameter, if **Parameter editor** is `Drop Down Selection` or `Check Box`, select a value from the **Default value** list. Otherwise, enter a value in the **Default value** field.
   - To close the **Edit Parameter** window after you have completed your changes, click **OK**.

What to do next

You can build the pattern plug-ins, see "Building pattern plug-ins" on page 2126.

Adding and editing parameter groups

Edit parameter groups to control how pattern parameters are displayed to pattern users when they create pattern instances from a user-defined pattern.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

After defining target properties, a default list of parameter groups is created. You can edit the default groups or add new groups.

Procedure

- To add a new group:
  a) In the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add Group**.
     The **Add Group** window opens.
  b) Enter a display name for the group in the **Display name** field.
  c) Enter a description for the group in the **Description** field.
  d) The **Generate help documentation** check box is selected by default. If you do not want to show the help documentation for the parameters in the group, clear the **Generate help documentation**

check box. To add help documentation for pattern parameters, see "Defining the user interface" on page 2074.

e) If you do not want the group to display with a surrounding box in the Configure Pattern Parameters page, clear the **Display parameters in a group box** check box.

If a group is displayed in a group box, pattern users can expand or collapse the group box to display or hide the parameters in that group. Parameters in a group that is not shown in a group box are always visible.

f) Click **OK**.

The new group is shown in the Groups and Parameters list.

- To edit an existing group:

a) In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the group name that you want to edit, or select the group name and click **Edit**.

The **Edit Group** window opens.

b) To change the display name, enter a display name for the group in the **Display name** field.

c) To change the description, enter a description for the group in the **Description** field.

d) If you want to show help documentation for the parameters in the group, ensure the **Generate help documentation** check box is selected. To add help documentation for pattern parameters, see "Defining the user interface" on page 2074.

e) If you want the group to display with a surrounding box in the Configure Pattern Parameters page, ensure the **Display parameters in a group box** check box is selected.

If a group is displayed in a group box, pattern users can expand or collapse the group box to display or hide the parameters in that group. Parameters in a group that is not shown in a group box are always visible.

f) Click **OK**.

The modified group is shown in the Groups and Parameters list.

What to do next

You can choose whether a parameter group is enabled based on the value of pattern parameters; see "Enabling parameter groups" on page 2077.

Enabling parameter groups

In a user-defined pattern, control whether parameter groups are enabled based on the values of pattern parameters.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

By default, when pattern users configure an instance of a user-defined pattern, all the parameter groups that are defined in the user-defined pattern are enabled. Enabled parameter groups and pattern parameters within those groups are visible to pattern users in the Configure Parameters page. You can control whether a parameter group in an instance of a user-defined pattern is enabled by using an XPath expression:

- If the expression evaluates to Boolean **true**, the parameter group is enabled. When a pattern user configures an instance of a user-defined pattern, the parameter group and pattern parameters within that group are visible to pattern users in the Configure Parameters page. The values of pattern

parameters within enabled parameter groups populate the target properties of the user-defined pattern.

- If the expression evaluates to Boolean `false`, the parameter group is disabled. When a pattern user configures an instance of a user-defined pattern, the parameter group is not shown to pattern users in the Configure Parameters page. Pattern parameters within disabled groups are not shown to pattern users and the values of those parameters do not populate the target properties in an instance of the user-defined pattern.

If no XPath expression is entered for a parameter group, the parameter group is enabled. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter group, or select a parameter group and click **Edit**. The **Edit Group** window opens.
2. Click the **Visibility** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table. To change parameter IDs, see "Changing pattern parameter IDs" on page 2081.

Procedure

To control whether a pattern parameter group is enabled by using an XPath expression:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter group that you want to control, or select the parameter group and click **Edit**.

   The **Edit Group** window opens.
2. Click the **Enable** tab. Create an XPath expression for your chosen parameter group:
   - To select a function:
     a. Expand **Boolean**, **Number**, **Pattern**, or **String** in the Functions section, then click a function. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:
     a. Click a pattern parameter in the Pattern Parameters table. The parameter ID shown in the **Parameter ID** column of the Pattern Parameters table for the chosen parameter is displayed in the **Parameter ID** field.

        You cannot select a pattern parameter that uses a table type.
     b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor. To change parameter IDs, see "Changing pattern parameter IDs" on page 2081.
   - You can also edit the expression directly in the **Expression** field.
3. Repeat the actions in step "2" on page 2078, as required, to create your XPath expression.
4. You can now choose whether to test your expression:
   - To test your expression, see "Testing a group enabling expression" on page 2079.
   - To accept the expression without testing it, click **OK**. The **Edit Group** window closes.

Testing a group enabling expression
In a user-defined pattern, test an XPath expression that is used for enabling parameter groups.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074
- "Enabling parameter groups" on page 2077

Procedure

To test your XPath expression:

1. If the **Enable** tab in the **Edit Group** window is not already open, complete the following steps to open the tab:
   a) In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter group containing the XPath expression that you want to test and click **Edit**.
      The **Edit Group** window opens.
   b) Click the **Enable** tab.
2. To test the expression, click **Evaluate**.
   The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section, and the result is shown in the **Result** field. The result shows whether the parameter group is enabled or disabled, and the value of the expression is shown in parentheses. If the value is `true`, the associated group is enabled; if the value is `false`, the group is disabled. Use the following table to determine the value of your XPath expression.

Result data type	Evaluates to true	Evaluates to false
Boolean	`true`	`false`
Numeric	Any non-zero value	`0` or `0.0`
String	Any string that returns `true` with a match that is not case-sensitive	Any string that does not return `true` with a match that is not case-sensitive

3. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**.
   The new test value is shown in the Pattern Parameters section.
4. Repeat steps "2" on page 2079 and "3" on page 2079, as required, to test the XPath expression.
   If you want to modify the expression, see "Enabling parameter groups" on page 2077.
5. When you have finished testing the expression, click **OK**.
   The **Edit Group** window closes.

Creating and configuring categories

Configure categories to which to assign user-defined patterns in the Patterns Explorer view.

Before you begin

Complete the following task:

- "Creating a pattern authoring project" on page 2069

About this task

You can drag your user-defined pattern into any existing category, or you can create a new category. If you create a category, you can write a category specification that provides information about the category for other users. User-defined patterns are shown within these categories in the Patterns Explorer view.

Procedure

To create a new category, update the category specification, assign your pattern to a category, rename a category, or remove a category, begin by opening the **Categories** tab of the Pattern Authoring editor. You can now choose one or more of the following options:

- To create a new category as a subcategory of an existing category:
  a) Select an existing category.
  b) Click **Add Category**.
     The **Add Category** window opens.
  c) Enter the name of your new category and click **OK**.
     The new category is added to the category list under the category previously selected. The **Save Your Pattern File** window opens, asking you to confirm whether you want to save your patterns file. Click **Yes**.
- To edit the category specification for a category:
  a) Double-click the category.
     The HTML editor opens containing the default category specification.
  b) Update the category specification to describe the category, and click **File** > **Save** to save any changes.
- To assign your user-defined pattern to a category, drag your user-defined pattern to your chosen category.
- To rename a category:
  a) Click **Rename Category**.
     The **Edit Category** window opens.
  b) Enter the new name, click **OK**.
     The **Save Your Pattern File** window opens, asking you to confirm whether you want to save your patterns file.
- To remove a category, click **Remove category**.
  The **Save Your Pattern File** window opens, asking you to confirm whether you want to save your patterns file. Click **Yes**.

Configuring SOAP nodes for user-defined patterns

Configure the target properties on SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes by using a WSDL file.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- “Creating a pattern authoring project” on page 2069

Procedure

To configure target properties on SOAP nodes by using a WSDL file:

1. Select an exemplar that uses a SOAPInput, SOAPRequest, or SOAPAsyncRequest node.
   See “Selecting the source files to use for a user-defined pattern” on page 2070.

2. Define the target properties and ensure that the `WSDL file name` target property is selected as a target property in the SOAPInput, SOAPRequest, or SOAPAsyncRequest node.

   See "Defining the target properties" on page 2074.

What to do next

When the pattern user generates an instance of the user-defined pattern, the WSDL file name is presented as a pattern parameter. After the pattern user inputs the WSDL file name, the properties in the WSDL file are used to configure the SOAP node target properties. Examples of WSDL properties are the port type, binding, and service port values.

The target properties are configured with the first occurrence of each WSDL property.

Changing pattern parameter IDs

Change default pattern parameter IDs to make it easier to write XPath, Java, and PHP code that uses parameter IDs.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

Pattern parameter IDs uniquely identify pattern parameters. As a pattern author, you use pattern parameter IDs when writing XPath expressions to transform or enable pattern parameters, or when writing Java or PHP code to modify pattern instances. Every pattern parameter in a user-defined pattern has a default pattern parameter ID assigned, but you can change pattern parameter IDs from the default values so that they are easier to identify in your code.

Procedure

To change pattern parameter IDs from the default values:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter for which you want to change the parameter ID, or select the parameter and click **Edit**.

   The **Edit Parameter** window opens.
2. In the **Parameter ID** field of the **Basic** tab, enter your new parameter ID.

   The new parameter ID can be made up of ASCII characters, but must start with a non-numeric character.
3. Click **OK**. The **Edit Parameter** window closes.

What to do next

- To use the new parameter ID in an XPath expression to transform pattern parameters, see "Transforming pattern parameters" on page 2084.
- To use the new parameter ID in an XPath expression to enable pattern parameters, see "Enabling pattern parameters" on page 2086.
- To use the new parameter ID in Java or PHP code to modify pattern instances, see "Modifying pattern instances by using Java or PHP" on page 2105.

Creating a project used in a user-defined pattern

Use the values of pattern parameters to control whether a project is created when a pattern user generates an instance of a user-defined pattern.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

You can control whether projects that are selected as source files in a user-defined pattern are created by using XPath expressions. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

- The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:
  1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter, or select a parameter and click **Edit**. The **Edit Parameter** window opens.
  2. Click the **Transform** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table.
- When a pattern instance is generated by a pattern user, transformation expressions are processed before any Java and PHP code that has been added to modify pattern instances. Transformation of pattern parameters is processed in the following sequence:
  1. Every parameter that has an XPath transformation expression is evaluated.
  2. The value of each parameter that has an XPath transformation expression is updated with the result of its evaluation, overwriting the value entered by the pattern user.
  3. The parameters are evaluated in top to bottom order, as they are listed in the **Pattern Configuration** tab of the Pattern Authoring editor.

Procedure

To control project creation by using an XPath expression:

1. Open the Pattern Authoring editor.
2. In the **Source** tab, in the Select Source Files section, double-click the project that you want to control, or select the project and click **Edit**.

   The **Edit Project** window opens.

3. Click the **Create** tab. Create an XPath expression for your chosen project:
   - To select a function:
     a. Expand **Boolean**, **Number**, **Pattern**, or **String** in the Functions section, then click a function. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:
     a. Click a pattern parameter in the Pattern Parameters table. The parameter ID shown in the **Parameter ID** column of the Pattern Parameters table for the chosen parameter is displayed in the **Parameter ID** field.

        You cannot select a pattern parameter that uses a table type.
     b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor.
   - You can also edit the expression directly in the **Expression** field.
4. Repeat the actions in step "3" on page 2083, as required, to create your XPath expression.
5. You can now choose whether to test your expression:
   - To test your expression, see "Testing a project creation expression" on page 2083.
   - To accept the expression without testing it, click **OK**. The **Edit Project** window closes.

Testing a project creation expression
Test your XPath project creation expression created in a user-defined pattern to check that it works correctly.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074
- "Creating a project used in a user-defined pattern" on page 2082

Procedure

To test your XPath expression:

1. If the **Create** tab in the **Edit Project** window is not already open, complete the following steps to open the tab:
   a) Open the Pattern Authoring editor.
   b) In the **Source Files** tab, in the Select Source Files section, double-click the project containing the XPath expression that you want to test, or select the project and click **Edit**.

      The **Edit Project** window opens.
   c) Click the **Create** tab.
2. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**.

   The new test value is shown in the Pattern Parameters section.

3. To test the expression, click **Evaluate**.

   The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section and the result is shown in the **Result** field. If the expression is not valid an error message is displayed in the **Result** field.
4. Repeat steps "2" on page 2083 and "3" on page 2084, as required, to test the XPath expression.

   If you want to modify the expression, see "Creating a project used in a user-defined pattern" on page 2082.
5. When you have finished testing the expression, click **OK**.

   The **Edit Project** window closes.

Transforming pattern parameters

Calculate a pattern parameter value from the values entered for other pattern parameters.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

You can transform pattern parameters by using XPath expressions; for example, to calculate a pattern parameter value from the values entered for other pattern parameters. IBM App Connect Enterprise supports XPath 1.0. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

- The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:
  1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter, or select a parameter and click **Edit**. The **Edit Parameter** window opens.
  2. Click the **Transform** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table.
- When a pattern instance is generated by a pattern user, transformation expressions are processed before any Java and PHP code that has been added to modify pattern instances. Transformation of pattern parameters is processed in the following sequence:
  1. Every parameter that has an XPath transformation expression is evaluated.
  2. The value of each parameter that has an XPath transformation expression is updated with the result of its evaluation, overwriting the value entered by the pattern user.
  3. The parameters are evaluated in top to bottom order, as they are listed in the **Pattern Configuration** tab of the Pattern Authoring editor.

Procedure

To transform a pattern parameter by using an XPath expression:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter that you want to transform, or select the parameter and click **Edit**. The **Edit Parameter** window opens.

   You cannot transform a parameter that uses a table type.

2. Click the **Transform** tab. Create an XPath expression for your chosen parameter:
   - To select a function:
     a. Expand **Boolean**, **Number**, **Pattern**, or **String** in the Functions section, then click a function. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:
     a. Click a pattern parameter in the Pattern Parameters table. The parameter ID shown in the **Parameter ID** column of the Pattern Parameters table for the chosen parameter is displayed in the **Parameter ID** field.

        You cannot select a pattern parameter that uses a table type.
     b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor.
   - You can also edit the expression directly in the **Expression** field.
3. Repeat the actions in step "2" on page 2085, as required, to create your XPath expression.
4. You can now choose whether to test your expression:
   - To test your expression, see "Testing a transformation expression" on page 2085.
   - To accept the expression without testing it, click **OK**. The **Edit Parameter** window closes.

Testing a transformation expression
Test your XPath transformation expression created in a user-defined pattern to check that it works correctly.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074
- "Transforming pattern parameters" on page 2084

Procedure

To test your XPath expression:

1. If the **Transform** tab in the **Edit Parameter** window is not already open, complete the following steps to open the tab:
   a) In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click the parameter containing the XPath expression that you want to test, or select the parameter and click **Edit**.

      The **Edit Parameter** window opens.
   b) Click the **Transform** tab.
2. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**.

   The new test value is shown in the Pattern Parameters section.
3. To test the expression, click **Evaluate**.

   The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section and the result is shown in the **Result** field. If the expression is not valid an error message is displayed in the **Result** field.

4. Repeat steps “2” on page 2085 and “3” on page 2085, as required, to test the XPath expression.
   If you want to modify the expression, see “Transforming pattern parameters” on page 2084.
5. When you have finished testing the expression, click **OK**.
   The **Edit Parameter** window closes.

Enabling pattern parameters

You can control whether pattern parameters can be edited by pattern users based on the values of other pattern parameters.

Before you begin

Complete the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Defining the target properties” on page 2074

About this task

By using XPath expressions you can control whether pattern users can edit pattern parameters in the Pattern Instance editor . If an expression evaluates to Boolean **true**, the pattern user can configure the parameter in the **Configure Pattern Parameters** page; if an expression evaluates to Boolean `false`, the parameter is read-only, and the pattern user cannot configure it. When the pattern user configures pattern parameters, expressions are evaluated whenever any of the pattern parameters used in an expression change.

IBM App Connect Enterprise supports XPath 1.0. For XPath reference information, including information about XPath functions, see W3C XPath 1.0 Specification.

The `pp:getValue()` function is included, in addition to the functions in the XPath 1.0 Specification. The `pp:getValue()` function takes the parameter ID of a pattern parameter and returns the value of that pattern parameter. To see the parameter ID for a pattern parameter:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, double-click a parameter, or select a parameter and click **Edit**. The **Edit Parameter** window opens.
2. Click the **Enable** tab. The parameter IDs for pattern parameters are shown in the **Parameter ID** column of the Pattern Parameters table.

Procedure

To define an XPath expression to control whether a pattern parameter can be edited:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter that you want to configure. Click **Edit**.
   The **Edit Parameter** window opens.

2. Click the **Enable** tab. Create an XPath expression for your selected parameter by applying the following operations as required:
   - To select a function:
     a. Click a function in the Functions section. The function is displayed in the **Function name** field.
     b. Click **Use**. The function is inserted into the **Expression** field at the cursor.
   - To select an operator:
     a. Click an operator in the Operators section. The operator is displayed in the **Operator** field.
     b. Click **Use**. The operator is inserted into the **Expression** field at the cursor.
   - To select a pattern parameter:
     a. Click a pattern parameter in the Pattern Parameters table. The ID shown in the **Parameter ID** column of the Pattern Parameters table for the selected parameter is displayed in the **Parameter ID** field.

        You cannot select a pattern parameter that uses a table type.
     b. Click **Use**. The parameter ID is inserted into the **Expression** field at the cursor.
   - You can also edit the expression directly in the **Expression** field.

   A common requirement is for a pattern parameter to be read-only. To ensure that a pattern parameter is read-only, set the XPath expression for the pattern parameter to `false()`.
3. You can now choose whether to test your expression:
   - To test your expression, see "Testing an enabling expression" on page 2087.
   - To accept the expression without testing it, click **OK**. The **Edit Parameter** window closes.

Testing an enabling expression
Test an XPath enabling expression that you created in a user-defined pattern to check that it works correctly.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074
- "Enabling pattern parameters" on page 2086

Procedure

To test your XPath expression:

1. If the **Enable** tab in the **Edit Parameter** window is not already open, complete the following steps to open the tab:
   a) In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter containing the XPath expression that you want to test and click **Edit**.

      The **Edit Parameter** window opens.
   b) Click the **Enable** tab.
2. To test the expression, click **Evaluate**.

   The expression is evaluated by using the values in the **Test Values** column of the Pattern Parameters section, and the result is shown in the **Result** field. The result shows whether the parameter is enabled or disabled, and the value of the expression is shown in parentheses. If the value is `true`, the

associated parameter is enabled; if the value is `false`, the parameter is disabled. Use the following table to determine the value of your XPath expression.

Result data type	Evaluates to True	Evaluates to False
Boolean	`true`	`false`
Numeric	Any non-zero value	`0` or `0.0`
String	Any string that returns `true` with a match that is not case-sensitive	Any string that does not return `true` with a match that is not case-sensitive

3. To change the test value for a parameter, select the parameter in the Pattern Parameters section and enter the required test value in the **Test value** field. Click **Set**.

   The new test value is shown in the Pattern Parameters section.
4. Repeat steps "2" on page 2087 and "3" on page 2088, as required, to test the XPath expression.

   If you want to modify the expression, see "Enabling pattern parameters" on page 2086.
5. When you have finished testing the expression, click **OK**.

   The **Edit Parameter** window closes.

Using enumerated values for pattern parameters

Use enumerated types to provide predefined values for the pattern parameters in your user-defined patterns.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074

About this task

You can use enumerated types to provide pattern users with a predefined list of pattern parameter values to select from. For enumerated types that you create, you can add, remove, and edit values. As part of the pattern authoring process, any target properties that use a list of values have enumerated types created for them automatically. You can remove values from this automatically generated list, but you cannot add new values or change existing values.

Procedure

- Create your own enumerated types.

  See "Creating enumerated types for pattern parameters" on page 2088.
- Edit enumerated types.

  See "Editing enumerated types for pattern parameters" on page 2089.
- Assign enumerated types to pattern parameters.

  See "Defining the user interface" on page 2074.

Creating enumerated types for pattern parameters

Define enumerated types to provide predefined values for pattern parameters in your user-defined patterns.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069

- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Defining the target properties” on page 2074

About this task

Enumerated types give pattern users a predefined list of values to select from when they choose values for pattern parameters after generating a new pattern instance.

Procedure

To create a new enumerated type:

1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Enumerated Types**.
   The **Configure Enumerated Types** window opens.
2. Click the **Add** button that is under the **Enumerated type** field.
   The **Enter New Enumeration** window opens.
3. Enter a name for the enumeration in the **Enter a name for the enumeration** field. Click **OK**.
   The new enumeration is displayed in the table of values, and a display name and default value are entered in the table.
4. For manipulating values, either adding or editing, use the following options:
   - To add a new value, click the **Add** button that is next to the table of values. A new display name and value are added to the table.
   - To edit a display name or value, double-click the display name or value in the table. Type a new entry and then press Enter to save the new display name or value.
5. When you have finished adding values and editing display names and values, click **OK**.
   The **Configure Enumerated Types** window closes.

What to do next

You can now assign your enumerated type to a pattern parameter, see “Defining the user interface” on page 2074.

Editing enumerated types for pattern parameters

Edit enumerated types to add, remove or rename values in existing enumerated types, or to rename an enumerated type. You can edit an existing user-generated enumerated type to add, remove, or change values and display names within the enumerated type, or to change the name of the enumerated type.

Before you begin

Complete the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Defining the target properties” on page 2074
- “Creating enumerated types for pattern parameters” on page 2088

About this task

As part of the pattern authoring process, target properties that use a list of values have enumerated types automatically created for them. You can remove values from this automatically generated list or regenerate the original list of values, but you cannot add new values or edit existing values.

You can duplicate or remove user-generated enumerated types and enumerated types created from target properties.

For enumerated types that you create, you can add, remove, and edit values.

Procedure

To edit enumerated types:

1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Enumerated Types**. The **Configure Enumerated Types** window opens.
2. In the **Enumerated type** field, select the enumerated type to edit.
3. You can make the following changes to user-generated enumerated types:
   - Rename the enumerated type. To rename the enumerated type, click **Rename**. The **Rename Enumeration** window opens. Enter a name for the enumeration and click **OK**. The **Rename Enumeration** window closes.
   - Add new values. To add a new value, click the **Add** button that is next to the table of values. A new value and display name are added to the table.
   - Remove values. To remove a value, select the value to remove and click the **Remove** button that is next to the table of values. The value is removed from the table.

     If a value is the only entry in the table, you cannot remove it.
   - Change existing values and display names. To edit a value or display name, double-click the value or display name in the table. Type a new entry and then press Enter to save the new display name or value.
   - Duplicate the enumerated type. To duplicate an enumerated type, click **Duplicate**. The duplicate enumerated type is displayed. The name of the duplicate enumerated type is the same as the original, but with _1 added to the end.
   - Remove the enumerated type. To remove an enumerated type, click **Remove**. The enumerated type is removed.

     If an enumerated type is being used by a pattern parameter it cannot be removed. You can check if an enumerated type is being used by a parameter in the **This enumerated type is used by the following parameter** field. To modify a parameter so that it does not use an enumerated type, see "Defining the user interface" on page 2074.
4. You can make the following changes to enumerated types that are created from target properties:
   - Rename the enumerated type. To rename the enumerated type, click **Rename**. The **Rename Enumeration** window opens. Enter a name for the enumeration and click **OK**. The **Rename Enumeration** window closes.
   - Remove values. To remove a value, select the value to remove and click the **Remove** button that is next to the table of values. The value is removed from the table.

     If a value is the only entry in the table, you cannot remove it.
   - Reset the original values. To restore values you have removed from the list, click **Reset Values**. The original list is displayed.
   - Duplicate the enumerated type. To duplicate an enumerated type, click **Duplicate**. The duplicate enumerated type is displayed. The name of the duplicate enumerated type is the same as the original, but with _1 added to the end.
   - Remove the enumerated type. To remove an enumerated type, click **Remove**. The enumerated type is removed.

     If an enumerated type is being used by a pattern parameter it cannot be removed. You can see if an enumerated type is being used by a parameter in the **This enumerated type is used by the following parameter** field. To modify a parameter so that it does not use an enumerated type, see "Defining the user interface" on page 2074.
5. Repeat the tasks in steps "3" on page 2090 and "4" on page 2090 until your changes are complete. Click **OK**.

   The **Configure Enumerated Types** window closes.

What to do next

You can now build your pattern; see “Building pattern plug-ins” on page 2126.

Using tables for pattern parameters

Use table parameter types to create tables of pattern parameters in your user-defined patterns.

Before you begin

Complete the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Defining the target properties” on page 2074

About this task

You can add tables to your user-defined patterns, into which pattern users can enter rows of data. To add a table, you must define a table parameter type by specifying the columns you want in the table. You must then assign this parameter type to a pattern parameter when you define the user interface for the user-defined pattern.

When a pattern user configures an instance of your user-defined pattern, the table columns are displayed and the pattern user can enter one or more rows of data into the table. You can write Java or PHP code to use the values entered in parameter tables to modify the pattern instance.

When you use tables in your user-defined patterns, consider the following points:

- Values entered in parameter tables are handled as strings.
- Values entered in parameter tables cannot be transformed by using XPath expressions and cannot be used in XPath expressions.
- You can use enabling XPath expressions to enable a pattern parameter that uses a parameter table type.

Procedure

- To create a table parameter type, see “Adding pattern parameter table types” on page 2091.
- To edit a table parameter type, see “Editing pattern parameter table types” on page 2092.
- To assign a table parameter type to a parameter, see “Defining the user interface” on page 2074.
- To write Java or PHP code to read the values entered in a parameter table, see “Reading table values” on page 463 or “Examples of PHP API code” on page 2123.

Adding pattern parameter table types

Add table parameter types to include tables of pattern parameters in your user-defined patterns.

Before you begin

Complete the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Defining the target properties” on page 2074

About this task

Create pattern parameter table types to add parameter tables to your user-defined patterns. To define a table parameter type you must specify the columns of the table. For every column, you must specify a display name and a column ID.

Procedure

To create a table parameter type:

1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Tables**.
   The **Configure Tables** window opens.
2. Click the **Add** button that is next to the **Table** field.
   The **Enter New Table** window opens.
3. Enter a name for the table in the **Enter a name for the table** field. Click **OK**.
   The new table name is displayed in the **Table** field.
4. Define the columns you want to include in the table:
   - To add a new column, click the **Add** button that is next to the table of column entries. A new display name and column ID are added to the table.
   - To edit a display name, column ID, default value, or width, double-click the entry you want to edit. Type a new entry, then press Enter to save the new value.
5. To make a table column mandatory, select the required column in the table of column entries and select **Mandatory column**.
6. To enter help text for a table column, select the required column in the table of column entries and type the text in the **Enter any HTML or text that you want to display as help text for this column** field.
7. Click **OK**.

What to do next

To add a table to your user-defined pattern you must assign your parameter table type to a pattern parameter; see "Defining the user interface" on page 2074.

Editing pattern parameter table types

Edit pattern parameter table types in your user-defined pattern to add, remove, rename, or duplicate the table type, or to add, remove, or edit columns in the table type.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Defining the target properties" on page 2074
- "Adding pattern parameter table types" on page 2091

About this task

Procedure

To edit pattern parameter table types:

1. In the Pattern Authoring editor, click the **Pattern Configuration** tab. Click **Tables**.
   The **Configure Tables** window opens.
2. In the **Table** field, select the table type you want to edit.

3. You can make the following changes to table types:
   - Remove the table type. To remove the table type, click **Remove**.

     If a table type is used by a pattern parameter it cannot be removed. You can check if a table type is used by a parameter in the **This table is used by the following parameters** field. To modify a parameter so that it does not use a table type, see "Defining the user interface" on page 2074.
   - Rename the table type. To rename the table type, click **Rename**. The **Rename Table** window opens. Enter a new name for the table and click **OK**.
   - Duplicate the table type. To duplicate the table type, click **Duplicate**. The duplicate table type is displayed. The name of the duplicate table type is the same as the original, but with _1 added to the end.
   - Add columns to your table type. To add a new column, click the **Add** button that is next to the table of column entries. A new display name and column ID are added to the table.
   - Remove columns from your table type. To remove a column, click the **Remove** button that is next to the table of column entries.

     If a column entry is the only entry in the table, you cannot remove it.
   - Edit the display name, column ID, default value, or width for a column in your table type. To edit a display name, column ID, default value, or width, double-click the entry you want to edit. Type a new entry, then press Enter to save the new value.
   - Specify that a column in your table type is mandatory. To make a table column mandatory, select the required column in the table of column entries and select **Mandatory column**.
   - Enter help text for a column in your table type. To enter help text for a table column, select the required column in the table of column entries and type the text in the **Enter any HTML or text that you want to display as help text for this column** field.
4. When you have finished editing your pattern parameter table type, click **OK**.

   The **Configure Tables** window closes.

What to do next

You can now build your pattern; see "Building pattern plug-ins" on page 2126.

Using user-defined editors for pattern parameters

Create and configure a user-defined editor, which is used by pattern users to enter the value for a pattern parameter in your user-defined pattern.

About this task

When you define the user interface for your user-defined pattern, you must assign an editor to every pattern parameter. In addition to the built-in editors, you can create user-defined editors. The appearance and function of a user-defined editor is defined by Java code that you must write.

Procedure

To use a user-defined editor for a pattern parameter in a user-defined pattern:

1. Select the user-defined editor type for the required pattern parameter when you define the user interface for your user-defined pattern; see "Defining the user interface" on page 2074.
2. Configure the user-defined editor; see "Configuring a user-defined editor" on page 2094. When you configure the user-defined editor, you must specify the Java class that runs when the editor is used. For more information about writing Java code for user-defined editors, see "Writing Java code for a user-defined editor" on page 2095.

Configuring a user-defined editor
Configure a user-defined editor to specify the Java code that the editor uses and to specify configuration values and notifications used by the code.

Before you begin

Assign a user-defined editor to a pattern parameter; see "Defining the user interface" on page 2074.

About this task

To configure a user-defined editor for a pattern parameter, select a Java code project and a Java class within that project. The code in the Java class is run when the editor is used by a pattern user. You can create a new code project and Java class if required.

You can use your code project for user-defined editor code and for Java and PHP code to modify pattern instances. For more information about using Java and PHP to modify pattern instances, see "Modifying pattern instances by using Java or PHP" on page 2105. By including user-defined editor and pattern instance modification code in the same code project for patterns that use both types of code, you need to distribute fewer projects to pattern users.

You can specify configuration values when you configure your user-defined editor. These values can be used by your user-defined editor code. For example, you can enter a list of configuration values that are used to populate a list in your user-defined editor.

You can select parameters in your user-defined pattern that send change notifications to the user-defined editor. Your editor code can use change notifications to update the value of the parameter to which the user-defined editor is assigned when other pattern parameter values change.

Procedure

To configure a user-defined editor, complete the following steps:

1. In the **Pattern Configuration** tab of the Pattern Authoring editor, select the parameter to which you have assigned a user-defined editor. Click **Edit**.

   The **Edit parameter** window opens.
2. Click the **Editor** tab. In the **Parameter editor** field, confirm that **User-Defined Editor** is selected, then click **Configure Editor**.
3. In the **Project name** field, select the project that contains the Java code for your user-defined editor. To create a new code project, click **New Project** and complete the following steps:
   a) Enter a plug-in ID for the new code plug-in.

      After the plug-in is created, the plug-in ID is shown in the **Projects** section of the Application Development view. The plug-in ID is also used to identify the plug-in when you distribute it to pattern users.
   b) Clear **Add an example pattern authoring Java class to the project**.
   c) Clear **Add PHP support to the project**.
   d) Click **Finish**.
4. In the **Class name** field, select the class in the code project you want to run when the user-defined editor is used.

5. Optional: You can create a new Java class by completing the following steps:
   a) Click **New Java Class**.
      The **New Pattern Authoring Property Editor** window opens.
   b) In the **Source folder** field, click **Browse** and select the folder in which to store the new Java class file.
   c) Optional: In the **Package** field, enter the name of the Java package for the new class.
      If you leave this field blank, the default package is used.
   d) Optional: To add an interface for the new Java class, click **Add**. The **Implemented Interfaces Selection** window opens. In the **Choose interfaces** field, enter the name of the interface that you want to add, select the interface in the **Matching items** list and click **OK**. The **Implemented Interfaces Selection** window closes.
   e) Optional: To remove an interface for the Java class, select the interface in the **Interfaces** list and click **Remove**.
   f) Optional: Click **Next** to view information about the Pattern Authoring Property Editor template, which is used to create the Java class.
   g) Click **Finish**.
6. In the **Configuration values** field, enter any values that you want to be available in the Java code for your user-defined editor.
   For example, if your editor displays a list of values to a pattern user, you can enter the list of values in this field. Your editor code can then use the configuration values to display them in the editor.
7. In the **Select the visible parameters which will send change notifications to this user-defined editor** field, select all parameters in your user-defined pattern from which you want change notifications to be sent.
8. Click **OK**.

What to do next

Write the Java code for your user-defined editor; see "Writing Java code for a user-defined editor" on page 2095.

Writing Java code for a user-defined editor
Write Java code to define the appearance and function of your user-defined editor.

Before you begin

Create a code project and a Java class. You can create a code project and an example Java class for a user-defined editor when you configure your user-defined editor; see "Configuring a user-defined editor" on page 2094.

Read the reference information about the Java interfaces required to write code for user-defined editors; see Java interfaces for user-defined editors.

About this task

Enter Java code for your user-defined editor in the Java class you want to use. You must assign the Java class and the code project to your user-defined editor when you configure the editor; see "Configuring a user-defined editor" on page 2094. Pattern users edit an instance of your user-defined pattern by using the pattern instance editor. The code for your user-defined editor passes values to and receives values from the pattern instance editor by using two Java interfaces, `PatternPropertyEditor` and `PatternPropertyEditorSite`. You must write code to override the methods in these interfaces.

If an exception is thrown by user-defined editor code, the exception is caught and displayed in the **Pattern Instance Editor Exception** window, which opens automatically.

Procedure

To write the code for your user-defined editor:

1. Write code for your main user-defined editor class. Be aware of the following points:
   - The user-defined editor main class is created when the pattern user opens an instance of your user-defined pattern.
   - Your user-defined editor must extend the `BasePatternPropertyEditor` super class.
   - The main class must provide a default public constructor with no arguments.
2. In your main user-defined editor class, write code to override the `configureEditor()` method. Use this method to set up your user-defined editor after it is created. Be aware of the following points:
   - The `configureEditor()` method takes a `PatternPropertyEditorSite` object, a boolean value, and a string value. These values are passed automatically from the pattern instance editor.
   - The `PatternPropertyEditorSite` object acts as the interface between your user-defined editor code and the pattern instance editor. You can access this object by using the `getSite()` method anywhere in your user-defined editor code.
   - The boolean value indicates whether the pattern parameter for the user-defined editor has been flagged as mandatory. If it has, your code must set a value for the pattern parameter.
   - The string value contains the configuration values that are entered when configuring the user-defined editor. If required, you can write code to parse this string to use the values in your editor code.
   - In your `configureEditor()` method, call `configureEditor()` on the superclass, and pass the `PatternPropertyEditorSite` object, boolean value, and string value that were passed to your `configureEditor()` method. For example:

     ```
     @Override
      public void configureEditor(PatternPropertyEditorSite site, boolean required, String
      configurationValues) {
         super.configureEditor(site, required, configurationValues);
      }
     ```
3. In your main user-defined editor class, write code to override the `createControls()` method. Use this method to define the controls of your user-defined editor. Be aware of the following points:
   - You can create the controls in the `createControls()` method or write one or more new classes for the controls and instantiate these new classes from the `createControls()` method.
   - You must call the `valueChanged()` method on the `PatternPropertySite` object when the value of your pattern parameter changes. This call notifies the pattern instance editor of the change, which then updates any XPath expressions or user-defined editors which use this pattern parameter. You can access the `PatternPropertySite` object for your user-defined editor by using the `getSite()` method, for example, `PatternPropertyEditorSite site = getSite()`.
   - In the `createControls()` method, you must initialize your controls before defining any listeners for those controls.
4. In your main user-defined editor class, write code to override the `setValue()` method. Use this method to set initial values in your user-defined editor, for example, to populate a text box with the default value of the pattern parameter. The `setValue()` method takes a string value that is the current value of the pattern parameter, which is passed automatically from the pattern instance editor. The value passed from the pattern instance editor is the default value of the parameter, or the value that was saved after the pattern instance is configured.
5. In your main user-defined editor class, write code to override the `getValue()` method. Use this method to provide the value of the pattern parameter to the pattern instance editor. The `getValue()` method is automatically called after the `valueChanged()` method is called.
6. In your main user-defined editor class, write code to override the `setEnabled()` method. Use this method to enable or disable your user-defined editor. The `setEnabled()` method is passed a boolean value from the pattern instance editor to indicate whether the pattern parameter is enabled or

disabled by an enabling XPath expression. When the parameter is enabled, the value is *true*; when the parameter is disabled the value is *false*.

7. Optional: In your main user-defined editor class, write code to override the `isValid()` method. Use this method to validate entries in your editor and return an error message to the pattern instance editor if required. The error message is displayed to pattern users in the pattern instance editor.
8. Optional: In your main user-defined editor class, write code to override the `notifyChanged()` method. Use this method to handle change notifications from other parameters in your user-defined pattern and change the value of your pattern parameter or the behavior of your editor, if required. The `notifyChanged()` method takes the parameter ID of the changed parameter and the new value of that parameter.

Example

See the following examples of Java code for user-defined editors:

- For an example of user-defined editor code that uses one class, see "User-defined editor code example that uses one Java class" on page 2097.
- For an example of user-defined editor code that uses two classes, see "User-defined editor code example that uses two Java classes" on page 2099.
- For an example of a user-defined editor that handles change notifications from pattern parameters, see "User-defined editor code example for handling change notifications" on page 2101.

What to do next

If you have not configured your user-defined editor, see "Configuring a user-defined editor" on page 2094.

User-defined editor code example that uses one Java class
Use the example Java code to create a user-defined editor to display a text box, by using one Java class.

About this task

The following example code shows you how to use one Java class to display a text box user-defined editor to a pattern user. The `MyEditor` class extends the `BasePatternPropertyEditor` class and is specified as the user-defined editor class in the user-defined editor configuration; see "Configuring a user-defined editor" on page 2094. An instance of the `MyEditor` class is created automatically when the pattern user opens the pattern instance editor.

1. The `configureEditor()` method is called automatically after the user-defined editor class is created. This method sets up the editor.
2. The `createControls()` method is called to create the user interface for the editor:
   a. The SWT toolkit object, `Composite`, is used for the user interface.
   b. A text box control is created and the SWT layout information for the text box is defined.
   c. A listener on the text box is defined, which calls the `valueChanged()` method of the PatternPropertyEditorSite object for the editor. This ensures that when the selected value in the text box is changed, change notifications are sent to any XPath expressions or editors that use the value of this parameter.
3. The `setValue()` method sets the initial value for the text box. In this example, the code filters out blank values. The code also filters out values that start with <, to remove watermark values. This assumes watermark values are enclosed in angle brackets.
4. The `getValue()` method reads the current value of the text box and returns the value to the pattern instance editor. The pattern instance editor uses this returned value as the value of the pattern parameter.
5. The `setEnabled()` method enables or disables the text box control, depending on whether the pattern parameter is enabled or disabled. The parameter can be enabled or disabled by using an XPath expression; see "Enabling pattern parameters" on page 2086. When the result of an XPath expression

enables or disables the parameter, a boolean value is passed from the pattern instance editor to the `setEnabled()` method. If the parameter is enabled, the boolean value is *true*; if the parameter is disabled the boolean value is *false*.

```
package com.your.company.domain.MyPattern.code;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Text;

import com.ibm.broker.config.appdev.patterns.ui.*;

public class MyEditor extends BasePatternPropertyEditor {

   private Text text;

   @Override
   public void configureEditor(PatternPropertyEditorSite site, boolean required, String
 configurationValues) {
      super.configureEditor(site, required, configurationValues);
 }

   @Override
   public void createControls(Object parent) {
      Composite composite = (Composite) parent;

      text = new Text(composite, SWT.BORDER);

      GridData textLayoutData = new GridData(GridData.FILL_HORIZONTAL);
      textLayoutData.horizontalIndent = 0;
      text.setLayoutData(textLayoutData);

      text.addListener(SWT.Modify, new Listener() {
         @Override
         public void handleEvent(Event event) {
            PatternPropertyEditorSite site = getSite();
            site.valueChanged();
         }
      });
   }

   @Override
   public void setValue(String value) {

      if (value != null) {
         if (value.startsWith("<") == false) {
            text.setText(value);
         }
      }
   }

   @Override
   public String getValue() {
      String value = text.getText();
      if (value.length() > 0) {
         return value;
      }
      return null;
   }

   @Override
   public void setEnabled(boolean enabled) {
      text.setEnabled(enabled);
   }
}
```

User-defined editor code example that uses two Java classes
Use the example Java code to create a user-defined editor to display a list of values by using two Java classes.

About this task

The following example code shows you how to use two Java classes to display a list of values to a pattern user.

The MyEditor class controls the function of the editor. The MyComposite class controls the appearance of the editor by using the SWT toolkit.

The example assumes that the following steps are completed:

- The values shown in the list in the user-defined editor are entered as configuration values in the **Configure User-Defined Editor** window. The values must be entered as a comma-separated string, for example, `Apple,Orange,Pear,Lemon`. For more information about entering configuration values, see "Configuring a user-defined editor" on page 2094.
- The `MyEditor` class is entered in the **Configure User-Defined Editor** window, in the **Class name** field. For more information about specifying the class name for your user-defined editor code, see "Configuring a user-defined editor" on page 2094.

MyEditor class

The `MyEditor` class extends the `BasePatternPropertyEditor` class. An instance of the `MyEditor` class is created automatically when the pattern user opens the pattern instance editor.

1. The `configureEditor()` method is called automatically after the `MyEditor` class is created. This method reads the configuration values string for the user-defined editor and stores each of the comma-separated values.
2. The `createControls()` method is called to create the user interface for the editor. The controls are defined in the `MyComposite` class.
3. The `isValid()` method is called automatically after the `valueChanged()` method is called. In this example, `isValid()` checks the current value of the pattern parameter and returns an error message if no value is selected. The error message is displayed to the pattern user in the pattern instance editor. If the parameter value is valid, the method returns `null`.
4. The `setValue()`, `getValue()`, and `setEnabled()` methods are defined in the `MyComposite` class:
   a. The `setValue()` method sets the initial value in the user-defined editor.
   b. The `getValue()` method returns the current value of the pattern parameter to the pattern instance editor.
   c. The `setEnabled()` method is called when the pattern parameter is enabled or disabled by an XPath expression.

```
package com.your.company.domain.MyPattern.code;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;

import com.ibm.broker.config.appdev.patterns.ui.BasePatternPropertyEditor;
import com.ibm.broker.config.appdev.patterns.ui.PatternPropertyEditorSite;

public class MyEditor extends BasePatternPropertyEditor {
   private MyComposite composite;
   private String configurationValues;
   private String[] items;

   @Override
   public void configureEditor(PatternPropertyEditorSite site, boolean required, String
 configurationValues) {
      super.configureEditor(site, required, configurationValues);
      this.configurationValues = configurationValues;
      this.items = configurationValues.split("\\,");
      if (this.items == null) {
         this.items = new String[] { };
      }
```

```
   }

   @Override
   public void createControls(Object parent) {
      Composite parentComposite = (Composite) parent;
      PatternPropertyEditorSite site = getSite();
      composite = new MyComposite(parentComposite, SWT.NONE, items, site);
   }

   @Override
   public String isValid() {
      String selection = getValue();
      if (selection != null) {
         return null;
      }
      return "Nothing currently selected..!";
   }

   @Override
   public void setValue(String value) {
      composite.setValue(value);
   }

   @Override
   public String getValue() {
      return composite.getValue();
   }

   @Override
   public void setEnabled(boolean enabled) {
      composite.setEnabled(enabled);
   }
}
```

MyComposite class

The `MyComposite` class creates the user interface controls for the user-defined editor. The `MyComposite` class extends the SWT toolkit `Composite` class.

1. The layout of the controls is set and a new list control is created.
2. The `valueChanged()` method is used in a listener on the list control. This ensures that when the selected value in the list is changed, change notifications are sent to any XPath expressions or editors that use the value of this parameter.
3. The `setValue()`, `getValue()`, and `setEnabled()` methods are defined in the `MyComposite` class, but are called from the `MyEditor` class:
   a. The `setValue()` method takes the value of the pattern parameter and updates the list control to show the selected value.
   b. The `getValue()` method retrieves the currently selected value from the list control and, if it is not blank, returns it to `MyEditor`.
   c. The `setEnabled()` method uses the boolean value passed to it to enable or disable the list control.

```
package com.your.company.domain.MyPattern.code;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.List;

import com.ibm.broker.config.appdev.patterns.ui.PatternPropertyEditorSite;

public class MyComposite extends Composite {
   private List list;
   private PatternPropertyEditorSite site;

   public MyComposite(Composite parent, int style, String[] items, final
 PatternPropertyEditorSite site) {
      super(parent, SWT.NONE);
      this.site = site;
```

```
      GridLayout gridLayout = new GridLayout(1, false);

      gridLayout.marginWidth = 1;
      gridLayout.marginHeight = 1;
      gridLayout.horizontalSpacing = 0;
      gridLayout.verticalSpacing = 0;
      setLayout(gridLayout);

      setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));

      list = new List(this, SWT.BORDER);
      list.setItems(items);
      GridData listLayoutData = new GridData(GridData.FILL_HORIZONTAL);
      listLayoutData.horizontalIndent = 0;
      list.setLayoutData(listLayoutData);

      list.addSelectionListener(new SelectionAdapter() {
         @Override
         public void widgetSelected(SelectionEvent event) {
            site.valueChanged();
         }
      });
   }

   public void setValue(String value) {
      if (value != null) {
         list.setSelection(new String[] { value });
      }
   }

   public String getValue() {
      String[] selection = list.getSelection();
      if (selection.length > 0) {
         return selection[0];
      }
      return null;
   }

   public void setEnabled(boolean enabled) {
      list.setEnabled(enabled);
   }
}
```

User-defined editor code example for handling change notifications
Use the example Java code to create a user-defined editor that updates the value of the parameter when there are changes to other pattern parameters.

About this task

The following example code shows you how to write user-defined editor code to handle change notifications from other pattern parameters. The user-defined editor, assigned to pattern parameter *months*, displays a list of the months of the year, one or more of which the pattern user can select by using check boxes. A second pattern parameter, *uncheck month*, displays a list of the months of the year in a drop-down list. When the pattern user selects a month in the *uncheck month* parameter, the user-defined editor code receives a change notification from *uncheck month* and that month is cleared in the list of months in the user-defined editor.

The `MyEditor` class controls the function of the editor. The `MyComposite` class controls the appearance of the editor by using methods and objects from the SWT toolkit.

The example assumes that the following steps are completed:

1. The pattern parameters, *months* and *uncheck month* are added to the user-defined pattern. To add pattern parameters, see “Defining the user interface” on page 2074.
2. An enumerated type is created that contains the months of the year. The enumerated type is assigned to pattern parameter, *uncheck month*. To use enumerated types, see “Using enumerated values for pattern parameters” on page 2088.
3. The pattern parameter, *months*, is configured to use a user-defined editor; see “Defining the user interface” on page 2074.

4. The user-defined editor for pattern parameter, *months*, is configured by completing the following steps. For more information about configuring a user-defined editor, see "Configuring a user-defined editor" on page 2094.
   a. The `MyEditor` class is entered in the **Configure User-Defined Editor** window, in the **Class name** field.
   b. The pattern parameter, *uncheck month*, is selected in the **Configure User-Defined Editor** window, in the **Select the visible parameters which will send change notifications to this user defined editor** field.

MyEditor class

The `MyEditor` class extends the `BasePatternPropertyEditor` class. An instance of the `MyEditor` class is created automatically when the pattern user opens the pattern instance editor.

1. The `configureEditor()` method is called automatically after the `MyEditor` class is created.
2. The `createControls()` method is called to create the user interface for the editor. The controls are defined in the `MyComposite` class.
3. The `isValid()` method is called automatically after the `valueChanged()` method is called. In this example, `isValid()` checks the current value of the pattern parameter and returns an error message if no value is selected. The error message is displayed to the pattern user in the pattern instance editor. If the parameter value is valid, the method returns `null`.
4. The `setValue()`, `getValue()`, and `setEnabled()` methods are defined in the `MyComposite` class:
   a. The `setValue()` method sets the initial value in the user-defined editor.
   b. The `getValue()` method returns the current value of the pattern parameter to the pattern instance editor.
   c. The `setEnabled()` method is called when the pattern parameter is enabled or disabled by an XPath expression.
5. The `notifyChanged()` method calls the `uncheckMonth()` method in the MyComposite class, passing the value of the parameter, *uncheck month*. The `notifyChanged()` method also takes the ID of the updated pattern parameter, but this is not used in this example.

```
package com.your.company.domain.MyPattern.code;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;

import com.ibm.broker.config.appdev.patterns.ui.BasePatternPropertyEditor;
import com.ibm.broker.config.appdev.patterns.ui.PatternPropertyEditorSite;

public class MyEditor extends BasePatternPropertyEditor {
   private MyComposite composite;

   @Override
   public void configureEditor(PatternPropertyEditorSite site, boolean required, String
 configurationValues) {
      super.configureEditor(site, required, configurationValues);
   }

   @Override
   public void createControls(Object parent) {
      Composite parentComposite = (Composite) parent;
      PatternPropertyEditorSite site = getSite();
      composite = new MyComposite(parentComposite, SWT.NONE, site);
   }

   @Override
   public String isValid() {
      String selection = getValue();
      if (selection != null) {
         return null;
      }
      return "Nothing currently selected..!";
   }

   @Override
```

```
    public void setValue(String value) {
       if (value != null) {
          composite.setValue(value);
       }
    }

    @Override
    public String getValue() {
       return composite.getValue();
    }

    @Override
    public void setEnabled(boolean enabled) {
       composite.setEnabled(enabled);
    }

    @Override
    public void notifyChanged(String parameterId, String value) {
       composite.uncheckMonth(value);
    }
}
```

MyComposite class

The `MyComposite` class creates the user interface controls for the user-defined editor. The `MyComposite` class extends the SWT toolkit `Composite` class.

1. The layout of the controls is set and a new table control is created.
2. The `valueChanged()` method is used in a listener on the table control. This ensures that when the selected values in the table are changed, change notifications are sent to any XPath expressions or editors that use the value of this parameter.
3. A table item is created for each month of the year.
4. The `setValue()`, `getValue()`, and `setEnabled()` methods are defined in the `MyComposite` class, but are called from the `MyEditor` class:
   a. The `setValue()` method takes the value of the pattern parameter and updates the table control to show the selected values. The value of the parameter is stored as an XML document in string format that contains the list of months and information about whether each month is selected. The `setValue()` method reads each month, whether it is selected, and updates the check boxes in the table.
   b. The `getValue()` method retrieves the currently selected values from the table in the user-defined editor. The list of months and the check box status of each month is stored in an XML document, which is saved as a string value. This string is returned to the `MyEditor` class and then to the pattern instance editor. The string is stored as the current value of the parameter.
   c. The `uncheckMonth()` method takes the value of the parameter, *uncheck month*, passed from the `MyEditor` class. The month is checked against the list of months in the table and when a match is found, the check box for that month is cleared in the user-defined editor.
   d. The `setEnabled()` method uses the boolean value passed to it to enable or disable the table control.

```
package com.your.company.domain.MyPattern.code;

import java.io.StringReader;
import java.io.StringWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Table;
```

```
import org.eclipse.swt.widgets.TableItem;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

import com.ibm.broker.config.appdev.patterns.ui.PatternPropertyEditorSite;

public class MyComposite extends Composite {
   private PatternPropertyEditorSite site;
   private Table table;

   public static String[] MONTHS = {
      "January", "February", "March", "April", "May", "June", "July",
         "August", "September", "October", "November", "December" };

   public MyComposite(Composite parent, int style, final PatternPropertyEditorSite site) {
      super(parent, SWT.NONE);
      this.site = site;

      GridLayout gridLayout = new GridLayout(1, false);
      setLayout(gridLayout);

      setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true));

      table = new Table(this, SWT.BORDER | SWT.CHECK);
      table.setLayoutData(new GridData(SWT.FILL, SWT.FILL, true, true, 1, 1));
      table.setLinesVisible(true);

      table.addSelectionListener(new SelectionAdapter() {
         @Override
         public void widgetSelected(SelectionEvent event) {
            site.valueChanged();
         }
      });

      for (String currentMonth : MONTHS) {
         TableItem currentItem = new TableItem(table, SWT.NONE);
         currentItem.setText(currentMonth);
      }
   }

   public void setValue(String value) {

      try {

         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
         DocumentBuilder builder = factory.newDocumentBuilder();
         InputSource inputSource = new InputSource(new StringReader(value));
         Document document = builder.parse(inputSource);
         NodeList listOfMonths = document.getDocumentElement().getChildNodes();

         for (int index = 0; index < listOfMonths.getLength(); index++) {
            Node monthNode = listOfMonths.item(index);
            String monthName = monthNode.getNodeName();
            boolean checked = Boolean.parseBoolean(monthNode.getTextContent());

            for (TableItem tableItem : table.getItems()) {
               String itemName = tableItem.getText();
               if (itemName.equals(monthName) == true) {
                  tableItem.setChecked(checked); break;
               }
            }
         }
      } catch (Exception exception) { }
   }

   public String getValue() {

      try {

         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
         DocumentBuilder builder = factory.newDocumentBuilder();
         Document document = builder.newDocument();
         Element rootElement = document.createElement("months");

         document.appendChild(rootElement);

         for (TableItem currentItem : table.getItems()) {
            String monthName = currentItem.getText();
            boolean isChecked = currentItem.getChecked();
```

```
            Element monthElement = document.createElement(monthName);
            monthElement.setTextContent(Boolean.toString(isChecked));
            rootElement.appendChild(monthElement);
         }

         TransformerFactory transformerFactory = TransformerFactory.newInstance();
         Transformer transformer = transformerFactory.newTransformer();
         transformer.setOutputProperty(OutputKeys.INDENT, "yes");

         StringWriter stringWriter = new StringWriter();
         StreamResult result = new StreamResult(stringWriter);
         DOMSource source = new DOMSource(document);
         transformer.transform(source, result);
         return stringWriter.toString();

      } catch (Exception exception) { }

      return null;
   }

   public void uncheckMonth(String monthName) {
      for (TableItem tableItem : table.getItems()) {
         String itemName = tableItem.getText();
         if (itemName.equals(monthName) == true) {
            tableItem.setChecked(false); break;
         }
      }
   }

   public void setEnabled(boolean enabled) {
      table.setEnabled(enabled);
   }
}
```

Modifying pattern instances by using Java or PHP

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

When you author user-defined patterns, you can include Java or PHP code to modify pattern instances. This code is run when the pattern user generates an instance of a user-defined pattern and can be used to carry out a number of actions on the pattern.

By using the IBM Integration API, you can access all the nodes, connections, and pattern parameters in a user-defined pattern. Examples of changes that you can make by using Java code are:

- Adding, removing, and copying nodes
- Changing connections between nodes
- Renaming nodes
- Changing pattern parameter values and user-defined property values
- Renaming message flows

By using the PHP API for user-defined patterns, you can access the pattern parameters and the pattern instance name of a user-defined pattern. Examples of changes that you can make by using PHP code are:

- Marking up ESQL files to change the operation of the ESQL file depending on the value of pattern parameters
- Taking values from an XML file and using them as pattern parameters
- Changing pattern parameter values
- Renaming message flows
- Running other PHP scripts or Java code

Do not use Java or PHP API code to set the `WSDL file name` property, or any of the properties set by the WSDL file, on a SOAPInput, SOAPRequest, or SOAPAsyncRequest node in your user-defined pattern. To set the `WSDL file name` property, see “Configuring SOAP nodes for user-defined patterns” on page 2080. To modify other properties on these nodes, select the properties as target properties when you

create your user-defined pattern. To select target properties, see “Defining the target properties” on page 2074.

Choosing between Java and PHP

IBM App Connect Enterprise contains a Java to PHP bridge, therefore you can complete some pattern authoring tasks by using either Java or PHP. You can choose which language to use based on the existing skills, assets, or libraries within your organization. The IBM App Connect Enterprise Toolkit includes code completion and a debugger for Java, which make Java a good choice for longer or more complex code. PHP written in the IBM App Connect Enterprise Toolkit can be changed and tested without relaunching the workbench, which makes PHP a good choice if you want to change your code and view the results quickly. If you do not have a preference between Java and PHP after considering the skills and assets of your organization and the testing approach, use Java for changes to the structure of the message flow and use PHP to modify text files.

Plug-in packaging

The code you write to modify pattern instances is contained in a separate plug-in from the user-defined pattern. When you create a pattern archive, your code plug-ins are automatically packaged with the generated pattern plug-ins. For more information about packaging user-defined patterns, see “Packaging and distributing pattern plug-ins” on page 2127.

Sequence of actions when a pattern user generates a pattern instance

When a pattern user generates an instance of a user-defined pattern, the following actions occur in sequence:

1. The pattern instance projects are created and all non-message flow files are copied into the workspace.
2. The message flows in the pattern plug-ins are loaded into memory.
3. Pattern parameters that are transformed by using XPath expressions are evaluated. If a pattern parameter is disabled by using an XPath expression, its parameter values are not changed. All target properties are set in all message flows within the user-defined pattern based on pattern parameter values.
4. The Java and PHP code targets are run in top-to-bottom order as they are listed in the Pattern Authoring editor.
5. The message flows are saved into the pattern instance projects.

Creating a code plug-in project
Create a Java and PHP plug-in project to contain pattern instance modification code.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070

About this task

You must create a code plug-in project to contain the Java and PHP pattern instance modification code that you write. You can distribute the Java and PHP code plug-in project in the pattern archive for the associated user-defined pattern to pattern users. See “Packaging and distributing pattern plug-ins” on page 2127.

Procedure

To create a Java and PHP code plug-in project:

1. Open the **Pattern Configuration** tab of the Pattern Authoring editor.
2. Click **Add**.

   The **Add Java or PHP Code** window opens.
3. Click **New Project**.

   The **New Pattern Authoring Java and PHP Project** window opens.
4. Enter a plug-in ID for the new code plug-in.

   After the plug-in is created, the plug-in ID is shown in the **Projects** section of the Application Development view. The plug-in ID is also used to identify the plug-in when you distribute it to pattern users.
5. Optional: If you want to include PHP code in the code plug-in, select **Add PHP support to the project**.

   If you select this option a `main.php` template and example PHP scripts are added to the code plug-in.
6. Optional: If you want to include a Java class in the code plug-in, complete the following steps:
   a) Select **Add an example pattern authoring Java class to the project**.
   b) Enter a package name for the Java class in the **Package name** field.
   c) Enter a class name for the Java class in the **Class name** field.
7. Click **Finish**.

   The **New Pattern Authoring Java and PHP Project** window closes.
8. You can now add Java or PHP code to a user-defined pattern:
   - To add Java code to a user-defined pattern, see "Modifying pattern instances by using the IBM Integration API" on page 2107.
   - To add PHP code to a user-defined pattern, see "Modifying pattern instances by using PHP" on page 2121.
   - To return to pattern authoring without adding code, in the **Add Java or PHP Code** window, click **Cancel**.

Modifying pattern instances by using the IBM Integration API
Add Java code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Creating a code plug-in project" on page 2106

About this task

You can add Java code to a code plug-in project so that it runs when a pattern instance is generated. To add the Java code, select the Java class that you want to run.

The Java class you select can be created in the following ways:

- If you added a Java class when you created the code plug-in, then a template Java class is created and added to the project. This class is based on the Pattern Authoring Class template and contains the basic structure required to function correctly.
- When you follow the steps in this topic to select a Java class, you can optionally create a new Java class. A class created in this way is based on the Pattern Authoring Class template and contains the basic structure required to function correctly.
- You can write your own Java class. This class must implement the `GeneratePatternInstanceTransform` interface.

Regardless of how the class is created, you must add your own code within the class to complete the steps you require when the pattern instance is generated. For examples of IBM Integration API code for completing common tasks, see "Developing applications using the IBM Integration API: Example code" on page 455. For reference information about the IBM Integration API, see IBM Integration API.

Procedure

To select a Java class to run when a pattern instance is generated:

1. If the **Add Java or PHP Code** window is not open, in the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add**.

   The **Add Java or PHP Code** window opens.
2. In the **Type of code** list, select Java.
3. In the **Class name** list, select the name of the class that you want to run when a pattern instance is generated.

   A class is shown in the **Class name** list if it is in the current workspace and if it implements the `GeneratePatternInstanceTransform` interface.
4. Optional: You can create a new Java class:
   a) Click **New Java Class**.

      The **New Pattern Authoring Java Class** window opens.
   b) In the **Source folder** field, click **Browse** and select the folder in which to store the new Java class file.
   c) Optional: In the **Package** field, enter the name of the Java package for the new class.

      If you leave this field blank, the default package is used.
   d) Optional: Select **Enclosing type** and enter an enclosing type in the **Enclosing type** field.
   e) Enter a name for the new Java class in the **Name** field.
   f) Optional: Change the Superclass for the new Java class in the **Superclass** field.
   g) Optional: To add an interface for the new Java class, click **Add**. The **Implemented Interfaces Selection** window opens. In the **Choose interfaces** field, enter the name of the interface that you want to add, select the interface in the **Matching items** list and click **OK**. The **Implemented Interfaces Selection** window closes.
   h) Optional: To remove an interface for the Java class, select the interface in the **Interfaces** list and click **Remove**.
   i) Optional: Click **Next** to view information about the Pattern Authoring Class template, which is used to create the Java class.
   j) Click **Finish**.
5. Click **OK**.

   The **Add Java or PHP Code** window closes and the Java class is shown in the Java and PHP Code section of the **Pattern Configuration** tab.
6. Optional: The Java and PHP code listed in the Java and PHP Code section of the **Pattern Configuration** tab runs from top to bottom when a pattern instance is generated. To change the order in which the code is run, select the entry you want to move in the **Java and PHP Code** section and click the **Up** or **Down** button to change the position of the entry in the list.

What to do next

In the Java class that you selected to run when the pattern instance is generated, you must now write the code to perform the steps you require. For examples of IBM Integration API code for completing common tasks, see "Developing applications using the IBM Integration API: Example code" on page 455. For information about the Java development views and editors within the IBM App Connect Enterprise Toolkit, see Java Development User Guide - Views and Editors.

Developing applications using the IBM Integration API: Example code
Examples of IBM Integration API code for common tasks, to help you write your own Java code to develop message flow applications or modify user-defined patterns.

About this task

The IBM Integration API uses the `com.ibm.broker.appdev.*` classes in the `IntegrationAPI.jar` file.

The following topics show examples of Java code that you can use to complete common tasks:

Procedure

Loading an existing message flow into memory
Use the IBM Integration API when developing message flow applications to load a message flow into memory. You can then access the message flow in your Java code.

You must load a message flow into memory to use it with the IBM Integration API methods within your Java code. To load a message flow into memory, create a `File` object and use the `read()` method of the `FlowRendererMSGFLOW`, which makes the message flow available for your Java code. The `read()` method takes the message flow project that contains the required message flow file and the relative path to the message flow file from this project.

The following example shows how to load a message flow that is in the `mqsi` directory:

```
import java.io.File;
import java.io.IOException;
import com.ibm.broker.MessageBrokerAPIException;
import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.FlowRendererMSGFLOW;

public class LoadMessageFlow {
    public static void main(String[] args) {
        File msgFlow = new File("../mqsi/main.msgflow");
        try {
            MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
        } catch (IOException e) {
            // Add your own code here
            e.printStackTrace();
        } catch (MessageBrokerAPIException e) {
            // Add your own code here
```

```
            e.printStackTrace();
        }
    }
}
```

Pattern authoring

When you create patterns with the pattern authoring tool, use the `getMessageFlow()` method of the `PatternInstanceManager` object, which is automatically passed to your Java code. Modify a pattern instance to load a message flow into memory and make it available to other IBM Integration API methods.

The following example shows how to load a message flow that is in the default schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject",
 "main.msgflow");
        if (mf1 != null) {
           // Message flow was found
        }
        else {
           // Message flow was not found
        }
    }
}
```

The following example shows how to load a message flow that is in the `mqsi` schema:

```
public class MyJava implements GeneratePatternInstanceTransform {
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) {
        MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "mqsi/
main.msgflow");
        if (mf1 != null) {
           // Message flow was found
        }
        else {
           // Message flow was not found
        }
    }
}
```

Renaming a node
Use the IBM Integration API when developing message flow applications to rename a node.

To rename a node, you must first access the required node. You can access the node by using the existing name of the node and the `getNodeByName()` method. You can then rename the node by using the `setNodeName()` method, which takes the new node name as a parameter.

The following example shows how to rename a node that is named `My Input Node`:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Node mqinNode = mf1.getNodeByName("My Input Node");
mqinNode.setNodeName("New Input Node");
```

Pattern authoring

The following example shows how to rename the previous example node for a pattern:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mqinNode.setNodeName("New Input Node");
```

Adding a node and a subflow node
Use the IBM Integration API when developing message flow applications to add a new node or subflow node to a message flow.

You can add a new built-in node, or a new subflow node to a message flow.

- When you add a subflow node by using the IBM Integration API, you must link the subflow node with the subflow message flow by using the `setSubFlow()` method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

```
sfNode.setSubFlow(sub1);
```

- If you want to set a node property on a subflow node, use the `addNodeProperty()` method to set the property before you link the node to the subflow by using the `setSubFlow()` method. If you use `addNodeProperty()` after `setSubFlow()`, the node property might not be stored.

The following example shows you how to add a new built-in node:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
mf1.addNode(mqinNode);
```

The following example shows you how to add a new subflow node to a message flow:

1. The new subflow node is added to the message flow held in object `mf1`.
2. The subflow node is linked to the subflow message flow by using the setSubFlow() method.
3. The subflow node name is set to `My Sub Flow Node`.
4. A new subflow node is created and assigned to object `sfNode`.

The subflow can be stored in a `.msgflow` format:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File subFlow = new File("subflow.msgflow");
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

The subflow can be stored in a `.subflow` format:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
sFile subFlow = new File("subflow.subflow");
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

Pattern authoring

The following example shows you how to add a new built-in node to a pattern instance:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
mf1.addNode(mqinNode);
```

The following example shows you how to add a new subflow node to a message flow:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = new SubFlowNode();
sfNode.setNodeName("My Sub Flow Node");
sfNode.setSubFlow(sub1);
mf1.addNode(sfNode);
```

Setting the position of a node
Use the IBM Integration API to set the position of a node on the canvas in the Application Development view.

Set the position of a node on the canvas by using the `setLocation()` method of the node object. The following example sets the position of a new MQOutput node to coordinates x=300 pixels, y=100 pixels:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQOutputNode mqoutNode = new MQOutputNode();
mqoutNode.setLocation(300, 100);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQOutputNode mqoutNode = new MQOutputNode();
mqoutNode.setLocation(300, 100);
```

Copying a node
Use the IBM Integration API to copy a node or subflow node to a message flow.

You can copy a built-in or subflow node by using the `clone()` method. In the following example, a new MQInput node *mqinNode* is created and properties on the node are set. A new MQInput node *mqinNode1* is then created by copying *mqinNode* by using the `clone()` method. When the node is copied, the node properties are also copied.

Example:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
MQInputNode mqinNode1 = (MQInputNode) mqinNode.clone();
mqinNode1.setNodeName("Copy of My Input Node");
mf1.addNode(mqinNode1);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
MQInputNode mqinNode1 = (MQInputNode) mqinNode.clone();
mqinNode1.setNodeName("Copy of My Input Node");
mf1.addNode(mqinNode1);
```

Removing a node
Use the IBM Integration API when developing message flow applications to remove a node from a message flow.

To remove a node, you must first get the required node from the message flow object. In the following example, the `getNodeByName()` method is used to get the required node from message flow object *mf1*. The node is then removed by using the `removeNode()` method. When a node is removed, any connections to or from the node are also removed.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Node mqinNode = mf1.getNodeByName("My Input Node");
mf1.removeNode(mqinNode);
```

Pattern authoring

The following example is the same as the previous, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Node mqinNode = mf1.getNodeByName("My Input Node");
mf1.removeNode(mqinNode);
```

Adding connections between nodes
Use the IBM Integration API when developing message flow applications to add connections between nodes.

Pattern authoring

You can connect both subflow and built-in nodes. The first example shows you how to connect two built-in nodes. The second example shows you how to connect a built-in node to a subflow node.

To use the terminals of a subflow node, you must link the subflow node with the subflow message flow by using the `setSubFlow()` method of the subflow node object. For example, if you have assigned your subflow message flow to message flow instance *sub1* and you have assigned your subflow node to subflow node instance *sfNode*, you must use the following statement to link the subflow node with the subflow message flow:

```
sfNode.setSubFlow(sub1);
```

The following example shows you how to connect two built-in nodes:

1. A MQInput node and a Collector node are created.
2. The `getInputTerminal()` method is used to create a dynamic Input terminal called *NEWIN* on the Collector node.
3. The Input terminal is connected to the Output terminal of the MQInput node by using the `connect()` method of the message flow object *mf1*.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
CollectorNode colNode = new CollectorNode();
colNode.getInputTerminal("NEWIN");
mf1.connect(mqinNode.OUTPUT_TERMINAL_OUT, colNode.getInputTerminal("NEWIN"));
```

To use a static terminal on a node, you must use the appropriate constant defined for it in the IBM Integration API. These constants are listed in the IBM Integration API javadoc, see IBM Integration API.

The following example shows you how to connect a subflow node to a built-in node. You must load the subflow message flow and link it to a subflow node. You must use the `getInputTerminal()`, `getInputTerminals()`, `getOutputTerminal()` or `getOutputTerminals()` method to access the terminal on the subflow node to which you want to connect. The example code completes the following steps:

1. The main message flow, `main.msgflow`, and a Compute node in the main message flow are loaded into memory.
2. The subflow message flow, subflow.msgflow, and the subflow node in the main message flow are loaded into memory.
3. The `setSubFlow()` method of the subflow node is used to link the subflow message flow *sub1* to the subflow node *sfNode*.
4. The `getOutputTerminal()` method is used to get the Process terminal of the subflow node. The `connect()` method of the message flow object is used to connect this terminal to the Input terminal of the Compute node.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
```

```
ComputeNode compNode = (ComputeNode)mf1.getNodeByName("My Compute Node");
File subFlow = new File("subflow.msgflow");
MessageFlow sub1 = FlowRendererMSGFLOW.read(subFlow);
SubFlowNode sfNode = (SubFlowNode)mf1.getNodeByName("My Subflow Node");
sfNode.setSubFlow(sub1);
mf1.connect(sfNode.getOutputTerminal("Process"), compNode.INPUT_TERMINAL_IN);
```

The following example is the same as the previous example for connecting two build-in nodes, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("INPUTQ");
CollectorNode colNode = new CollectorNode();
colNode.getInputTerminal("NEWIN");
mf1.connect(mqinNode.OUTPUT_TERMINAL_OUT, colNode.getInputTerminal("NEWIN"));
```

The following example is the same as the previous example for using static terminals on a node, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ComputeNode compNode = (ComputeNode)mf1.getNodeByName("My Compute Node");
MessageFlow sub1 = patternInstanceManager.getMessageFlow("MyFlowProject", "subflow.msgflow");
SubFlowNode sfNode = (SubFlowNode)mf1.getNodeByName("My Subflow Node");
sfNode.setSubFlow(sub1);
mf1.connect(sfNode.getOutputTerminal("Process"), compNode.INPUT_TERMINAL_IN);
```

Adding and connecting user-defined nodes
Use the IBM Integration API to add user-defined nodes and to connect user-defined nodes to other nodes.

The code required to add and connect user-defined node is different to the code required for built-in nodes and subflow nodes.

When you write Java code for user-defined nodes you must be aware of the following information:

- User-defined nodes are supported by instances of the `GenericNode` class. To add user-defined nodes to message flows, create instances of `GenericNode` and add them to the message flow instance.
- To retrieve existing instances of a user-defined node, call `getNodeByName()` and cast the returned object to a `GenericNode` object.
- The terminals defined on your user-defined nodes are not automatically available in the API. If you create an instance of a `GenericNode` class, it does not have any input or output terminals listed. The methods `getInputTerminals()` and `getOutputTerminals()` return empty lists.
- To get an input terminal for a `GenericNode`, call `getInputTerminal()` and pass the terminal name that exists on the generic node. This method returns the input terminal and makes it available in the message flow object that contains your generic node. After you have used `getInputTerminal()` with a known terminal name, this input terminal is returned if `getInputTerminals()` is used.
- To get an output terminal for a `GenericNode`, call `getOutputTerminal()` and pass the terminal name that exists on the generic node. This method returns the output terminal and makes it available in the message flow object that contains your generic node. After you have used `getOutputTerminal()` with a known terminal name, this output terminal is returned if `getOutputTerminals()` is used.

The following example shows how you can add a user-defined node to a message flow and connect it to a built-in node:

1. An MQInput node is created and added to the message flow.
2. A user-defined node is created by using the `GenericNode` class and is added to the message flow object.
3. The static output terminal of the MQInput is assigned to the variable *outputTerminal*.
4. The input terminal of the user-defined node is assigned to the variable *inputTerminal* by using the `getInputTerminal()` method with the known terminal name *In*.
5. The nodes are connected by using the `connect()` method.

6. The final section of code shows that the input node is now available for use in the message flow, by using the `getInputTerminals()` method of the user-defined node instance.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);

MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("IN");
mf1.addNode(mqinNode);

GenericNode myNode = new GenericNode("MyUserDefinedNode");
myNode.setNodeName("MyNode");
mf1.addNode(myNode);

OutputTerminal outputTerminal = mqinNode.OUTPUT_TERMINAL_OUT;
InputTerminal inputTerminal = myNode.getInputTerminal("In");
mf1.connect(outputTerminal, inputTerminal);

InputTerminal[] inputTerminals = myNode.getInputTerminals();
System.out.println("Input terminals on my node:");
for (int i = 0; i < inputTerminals.length; i++) {
   InputTerminal inputTerminal = inputTerminals[i];
   System.out.println(inputTerminal.getName());
}
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");

MQInputNode mqinNode = new MQInputNode();
mqinNode.setNodeName("My Input Node");
mqinNode.setQueueName("IN");
mf1.addNode(mqinNode);

GenericNode myNode = new GenericNode("MyUserDefinedNode");
myNode.setNodeName("MyNode");
mf1.addNode(myNode);

OutputTerminal outputTerminal = mqinNode.OUTPUT_TERMINAL_OUT;
InputTerminal inputTerminal = myNode.getInputTerminal("In");
mf1.connect(outputTerminal, inputTerminal);

InputTerminal[] inputTerminals = myNode.getInputTerminals();
System.out.println("Input terminals on my node:");
for (int i = 0; i < inputTerminals.length; i++) {
   InputTerminal inputTerminal = inputTerminals[i];
   System.out.println(inputTerminal.getName());
}
```

Removing connections between nodes

Use the IBM Integration API to remove connections between nodes.

You can disconnect two nodes by using the `disconnect()` method of the message flow object. You must provide this method with the names of the terminal instances that you want to disconnect.

The following example shows how you can disconnect two nodes:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
MQInputNode mqinNode = (MQInputNode)mf1.getNodeByName("My Input Node");
MQOutputNode mqoutNode = (MQOutputNode)mf1.getNodeByName("My Output Node");
mf1.disconnect(mqinNode.OUTPUT_TERMINAL_OUT, mqoutNode.INPUT_TERMINAL_IN);
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
MQInputNode mqinNode = (MQInputNode)mf1.getNodeByName("My Input Node");
```

```
MQOutputNode mqoutNode = (MQOutputNode)mf1.getNodeByName("My Output Node");
mf1.disconnect(mqinNode.OUTPUT_TERMINAL_OUT, mqoutNode.INPUT_TERMINAL_IN);
```

Creating or changing user-defined properties
Use the IBM Integration API to create or change user-defined properties (UDPs).

You can create new UDPs and add them to the message flow, or you can discover existing UDPs and modify them.

The following example shows you how to create a UDP and add it to a message flow:

1. A UDP called `Property1` is created in parameter group `Group1`. The data type of the UDP is defined as a string, and the UDP is given the default value `Hello World!`
2. The UDP is then added to the message flow by using the `addFlowProperty()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
UserDefinedProperty udp = new UserDefinedProperty("Group1", "Property1",
 UserDefinedProperty.Usage.MANDATORY, UserDefinedProperty.Type.STRING, "Hello World!");
mf1.addFlowProperty(udp);
```

In the following example, the existing UDPs in a message flow are discovered by using the `getFlowProperties()` method on the message flow. The `setName()` method is then used to set the name of the first UDP to `Property3`:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
Vector<FlowProperty> flowProperties = mf1.getFlowProperties();
flowProperties.get(0).setName("Property3");
```

You can also modify user-defined properties by using the administration REST API, as described in "Setting message flow user-defined properties at run time by using the administration REST API" on page 436.

Pattern authoring

The following examples are the same as the previous UDP examples, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
UserDefinedProperty udp = new UserDefinedProperty("Group1", "Property1",
 UserDefinedProperty.Usage.MANDATORY, UserDefinedProperty.Type.STRING, "Hello World!");
mf1.addFlowProperty(udp);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
Vector<FlowProperty> flowProperties = mf1.getFlowProperties();
flowProperties.get(0).setName("Property3");
```

Changing pattern parameter values
Use the IBM Integration API to modify a pattern instance to change pattern parameter values.

You can change a pattern parameter value by using the `PatternInstanceManager` object.

The following example sets the value of the pattern parameter with the pattern parameter ID pp1 to `true`:

```
patternInstanceManager.setParameterValue("pp1", "true");
```

Reading table values
Use the IBM Integration API to read table values by accessing and extracting table values.

To read table values, you must first access the table and then extract the data from each row. To read the table values, use one of the following examples:

Use this example to access and extract the table values by providing the name of the columns. In this example, *noRows* is the number of rows in the table, *value* is the parameter that stores the table value, *column* is the name of the column in the table, and *pp3* is the parameter ID.

```
PatternParameterTable paramtable = pim.getParameterTable("pp3");
int noRows = paramtable.getRowCount();
String value;
PatternParameterRow row;
for(int i=0; i<noRows; i++) {
    row = paramtable.getRow(i);
    value = row.getValue("column");
    //insert your code here
}
```

Alternatively, if you do not provide the name of the columns, you can use this example to access and extract the table values by using the `getColumns()` method. In this example, *noColumns* is the number of columns in the table, *value* is the parameter that stores the table value, *columns* is an array containing the names of the columns in the table, and *pp3* is the parameter ID.

```
PatternParameterTable paramtable = pim.getParameterTable("pp3");
int noRows = paramtable.getRowCount();
PatternParameterRow row;
String[] columns;
String value;
for(int i=0; i<noRows; i++) {
    row = paramtable.getRow(i);
    columns = row.getColumns();
    int noColumns = columns.length;
    for(int j=0; j<noColumns; j++) {
        value = row.getValue(columns[j]);
        //insert your code here
    }
```

Creating ESQL modules

Use the IBM Integration API when developing message flow applications to create ESQL modules. You can then associate ESQL modules with nodes that use ESQL. For example, if you create a Compute node, you can write Java code to create an ESQL module and then associate that module with the node.

Examples of nodes that use ESQL are Compute, Database, DatabaseInput, and Filter nodes. If you are using a non-default broker schema, you must set the schema by using the `setBrokerSchema()` method.

The following example shows you how to create an ESQL module in the `mqsi` schema. The module is then assigned to a new Compute node by using the `setComputeExpression()` method:

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
ESQLModule module = new ESQLModule();
module.setBrokerSchema("mqsi");
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

The following example shows you how to create an ESQL module in the default schema. The `setBrokerSchema()` method is not required.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

The following example shows you how to discover an ESQL module from within an ESQL file by using the `getEsqlModules()` method. You can then use the ESQL module to set the compute expression on a Compute node by using the `setComputeExpression()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

Pattern authoring

The following examples are the same as the three previous examples, but tailored for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setBrokerSchema("mqsi");
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
ESQLModule module = new ESQLModule();
module.setEsqlMain("MyESQLMain");
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(module);
mf1.addNode(compNode);
```

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
File esql = new File("FileBatchProcessingSample_Branch.esql");
ESQLFile esqlFile = new ESQLFile(esql);
Vector<ESQLModule> esqlModules = esqlFile.getEsqlModules();
ComputeNode compNode = new ComputeNode();
compNode.setNodeName("My Compute Node");
compNode.setComputeExpression(esqlModules.get(0));
mf1.addNode(compNode);
```

Renaming a message flow

Use the IBM Integration API when developing message flow applications to rename message flows.

You can write code to rename a message flow by using the `setName()` method. In the following example, a message flow file called `main.msgflow` is renamed to `mainGenerated.msgflow`.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
mf1.setName(mf1.getName()+"Generated");
```

Pattern authoring

The following example is the same as the previous example, but for pattern authoring:

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
mf1.setName(mf1.getName()+"Generated");
```

Updating filter tables on Route nodes

Use the IBM Integration API to update filter tables on Route nodes.

You can add and update rows on the filter table of a Route node.

Adding a new row

The following example shows you how to add a new row to a filter table by using the `createRow()` method:

1. The message flow and the Route node are loaded into memory.
2. The filter table of the Route node is loaded into memory by using the `getFilterTable()` method of the RouteNode object.
3. A new filter table row is created by using the `createRow()` method.
4. The value of the `filter pattern` property on this new row is set to `value="123"` by using the `setFilterPattern()` method.
5. The `routing output terminal` property is set to NEWOUT by using the `setRoutingOutputTerminal()` method.
6. The new row is then added to the filter table by using the `addRow()` method.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
RouteNode.FilterTableRow newRow = filterTable.createRow();
newRow.setFilterPattern("value=\"123\"");
newRow.setRoutingOutputTerminal("NEWOUT");
filterTable.addRow(newRow);
```

Updating a row

The following example shows you how to update rows on the filter table of a Route node.

1. The message flow, Route node, and filter table of the Route node are loaded into memory.
2. The rows of the filter table are loaded into memory by using the `getRows()` method.
3. The `filter pattern` property of the first row of the filter table is set to `value2="456"`.
4. The `routing output terminal` property of the first row of the filter table is set to NEWOUT2.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
Vector<RouteNode.FilterTableRow> filterTableRows = filterTable.getRows();
filterTableRows.get(0).setFilterPattern("value2=\"456\"");
filterTableRows.get(0).setRoutingOutputTerminal("NEWOUT2");
```

Pattern authoring

The following examples are the same as the previously described examples, but for pattern authoring:

Adding a new row

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
RouteNode.FilterTableRow newRow = filterTable.createRow();
newRow.setFilterPattern("value=\"123\"");
newRow.setRoutingOutputTerminal("NEWOUT");
filterTable.addRow(newRow);
```

Updating a row

```
MessageFlow mf1 = patternInstanceManager.getMessageFlow("MyFlowProject", "main.msgflow");
RouteNode routeNode = (RouteNode)mf1.getNodeByName("My Route Node");
RouteNode.FilterTable filterTable = (RouteNode.FilterTable)routeNode.getFilterTable();
Vector<RouteNode.FilterTableRow> filterTableRows = filterTable.getRows();
filterTableRows.get(0).setFilterPattern("value2=\"456\"");
filterTableRows.get(0).setRoutingOutputTerminal("NEWOUT2");
```

Running PHP code
Use the IBM Integration API to modify a pattern instance to run PHP scripts from within your Java code.

You can run PHP scripts from within the IBM Integration API by using the `runScript()` method of the PatternInstanceManager object. In the following example, the IBM Integration API runs a PHP script in the `com.your.company.domain.code` plug-in, `templates/script.php`.

```
@Override
public void onGeneratePatternInstance (PatternInstanceManager patternInstanceManager) {
 patternInstanceManager.runScript("com.your.company.domain.code", "templates/script.php");
}
```

Saving a message flow file from memory
Use the IBM Integration API when developing message flow applications to save a message flow from memory.

Save a message flow to a message flow file to preserve additions and modifications that you have made. To save a message flow from memory, use the `write()` method of the `FlowRendererMSGFLOW` object. The `write()` method writes all the changes made using the IBM Integration API to the specified message flow file.

The schema for a message flow is stored in the message flow, and cannot be modified by using the IBM Integration API. When you save your message flow file by using the IBM Integration API, the file is saved to a directory that has the same name as the schema. If the directory does not exist, it is created. For example, if your message flow is part of the `mqsi` schema, when you save the message flow file it is saved to the `mqsi` directory.

The following example shows how to save a message flow in the specified directory by using the `write()` method. On Windows, if the message flow is part of the `mqsi` schema, the message flow file is saved to `C:\MB\mqsi\main.msgflow`.

```
File msgFlow = new File("main.msgflow");
MessageFlow mf1 = FlowRendererMSGFLOW.read(msgFlow);
FlowRendererMSGFLOW.write(mf1,"C:\MB");
```

Testing Java code
After writing Java code to modify pattern instances, test the code to check that it works correctly.

Before you begin

Before you test your Java code, you must build the pattern plug-ins; see “Building pattern plug-ins” on page 2126.

About this task

You can test your Java code either by creating an instance of a pattern and checking that it is modified as you expect or by using the Java debugger to step through the code.

Procedure

- To test your user-defined pattern by creating an instance of the pattern, see “Testing a user-defined pattern” on page 2126.

- To use the Java debugger, you must open a temporary workspace to generate an instance of your user-defined pattern, then return to the original workspace to step through the code. To use the Java debugger, complete the following steps:
  a) In your Java code file, add a breakpoint to the required line of code. To add a breakpoint, right-click the marker bar (vertical ruler) to the left of the required code and click **Toggle Breakpoint**.
  b) In the Pattern Authoring editor, click the **Create Pattern** tab.
  c) Click **Debug Pattern**.
     The **Workspace Launcher** window opens.
  d) Select the temporary workspace in which to generate an instance of your user-defined pattern. Click **Browse** and select a workspace or enter a workspace in the **Workspace** field. Click **OK**.
     The selected workspace opens.
  e) In the temporary workspace, you must generate an instance of your user-defined pattern. In the Application Development view, click the **Patterns Explorer** tab and select your user-defined pattern.
  f) In the **Pattern Specification** tab, click **Create New Instance**. Enter a name for the instance of your user-defined pattern and click **OK**.
  g) Enter any mandatory pattern parameters and click **Generate**.
     The generation of the pattern instance stops when it reaches the breakpoint in your Java code.
  h) In the original workspace, in the **Confirm Perspective Switch** window, click **Yes**.
     The original workspace switches to debug mode and shows the code paused at your breakpoint.
  i) You can now use the Java debugger to test your code; see "Java Debugger" on page 664.
  j) When you have finished testing the code in the Java debugger, close the temporary workspace and switch back to the Integration Development perspective in the original workspace; see perspectives.

What to do next

You can either change your Java code and retest it or you can package and distribute your user-defined pattern:

- To change and retest your Java code, edit the code file and then repeat the steps in this topic.
- To package and distribute your user-defined pattern, see "Packaging and distributing pattern plug-ins" on page 2127

To ensure that changes in your Java code are included in your generated pattern, change the version number of the Java plug-in, as described in the following steps.

1. Update the version number of the Java plug-in in `manifest.mf`.
2. Update the version number of the pattern on the **Create Pattern** tab.
3. Re-create the pattern plug-ins and pattern archive, as described in "Building pattern plug-ins" on page 2126.
   To check that the pattern archive contains the correct versions of the plug-ins, you can look at the `feature.xml` file.
4. Install the pattern.

Modifying pattern instances by using PHP

Add PHP code to a code plug-in project to modify a pattern instance when the pattern instance is generated by a pattern user.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 2069

- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Creating a code plug-in project” on page 2106

About this task

You can add PHP code to a code plug-in project by selecting a PHP script, which runs when a pattern instance is generated. You can choose whether to write the output of this script to a file. For example, you can write a PHP script that creates an ESQL file that is used in your user-defined pattern.

If you added PHP support when you created the code plug-in, a PHP template and example scripts are created and added to the project. Alternatively, you can write your own PHP script. Regardless of how the script is created, you must add your own code within the script to complete the steps you require when the pattern instance is generated. For examples of PHP code for completing common tasks, see “Examples of PHP API code” on page 2123. For reference information about the PHP pattern authoring API, see PHP API for user-defined patterns.

Procedure

To select a PHP script to run when a pattern instance is generated:

1. If the **Add Java or PHP Code** window is not open, in the **Pattern Configuration** tab of the Pattern Authoring editor, click **Add**.

   The **Add Java or PHP Code** window opens.
2. In the **Type of code** list, select PHP.
3. In the **Project name** list, select the project or plug-in that contains the PHP script that you want to run when a pattern instance is generated. If the required plug-in is installed, but is not in your workspace, click **Browse** and select the required plug-in.
4. In the **PHP file name** list, select the name of the script you want to run when a pattern instance is generated.

   The **PHP file name** list shows all the PHP scripts in the project or plug-in that you selected.
5. Optional: To write the output of the PHP script to a file, select **Write the output from the PHP file into an output file**.
   a) In the **Pattern instance project** list, select the pattern instance project in which you want to include the output file.
   b) In the **Output file name** field, enter a name for the output file.

      You can include a path in this field, for example `scripts/example.mqsc` to write the output to the `scripts` folder.
6. Click **OK**.

   The **Add Java or PHP Code** window closes and the PHP script that you selected is shown in the **Java and PHP Code** section of the **Pattern Configuration** tab.
7. Optional: The Java and PHP code listed in the Java and PHP Code section of the **Pattern Configuration** tab runs from top to bottom when a pattern instance is generated. To change the order in which the code runs, select the entry you want to move in the Java and PHP Code section and click the **Up** or **Down** button to change the position of the entry in the list.

What to do next

In the PHP script that you selected to run when the pattern instance is generated, you must now write the code to perform the steps you require. For examples of PHP code for completing common tasks, see “Modifying pattern instances by using PHP” on page 2121. For reference information about the PHP pattern authoring API, see PHP API for user-defined patterns.

Examples of PHP API code
Use the following examples of PHP API code for common tasks to help you write your own PHP code to modify pattern instances.

About this task

The following examples show PHP code that you can use to complete common tasks when you are creating user-defined patterns:

Procedure

- Running additional PHP scripts.
  The following example shows how to use a PHP script to run another PHP script, by using the `mb_pattern_run_template()` function. The `mb_pattern_run_template()` function uses the following parameters:

  1. The first parameter required by `mb_pattern_run_template()` is the name of the project containing the PHP template; `Transform` in this example.
  2. The second parameter required by `mb_pattern_run_template()` is the name of the PHP script to run; in this example `example.esql.php`, which is in a subfolder called `mqsi`.
  3. The third parameter required by `mb_pattern_run_template()` is the name of the file to which the output is written; in this example `example.esql`, in subfolder `mqsi`.

  The user-defined pattern in this example contains a Boolean pattern parameter called `includeErrorHandling`. This pattern parameter is used to create a check box in the pattern:

```
<?php
    if ($_MB['PP']['includeErrorHandling'] == 'true') {
        mb_pattern_run_template("Transform", "mqsi/example.esql.php", "mqsi/example.esql");
    }
?>
```

- Using PHP scripts with markup for ESQL.
  In the following example, the user-defined pattern contains a pattern parameter called `errorQueue`, which can contain the values none or `errorAction`:

```
BROKER SCHEMA mqsi
<?php
    if ($_MB['PP']['errorAction'] == 'errorQueue') {
        echo "DECLARE ErrorAction EXTERNAL CHARACTER '".$_MB['PP']['errorAction']."';";

        echo <<<ESQL

        CREATE FILTER MODULE CheckErrorAction
            CREATE FUNCTION Main() RETURNS BOOLEAN
            BEGIN
                IF ErrorAction = 'errorQueue' THEN
                    RETURN TRUE;
                ELSE
                    RETURN FALSE;
                END IF;
            END;
        END MODULE;

ESQL;
    }
?>
```

- Using PHP scripts to remove files from a project. In the following example, the script first checks the value of the check box pattern parameter, which has the pattern parameter ID pp1. If this parameter is set to `false`, the script deletes the `Log.msgflow` and `Log.esql` files from the pattern instance project. Message flow files must be deleted by using the `removeMessageFlow()` function. Non-message flow files can be deleted by using the standard PHP function `unlink()`:

```
if ($_MB['PP']['pp1'] == 'false') {

    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
```

```
    $logmsgflow = $pim->getMessageFlow("Example_Flows", "mqsi/Log.msgflow");
    $pim->removeMessageFlow($logmsgflow);

    $piworkspace = $pim->getWorkspaceLocation();
    $piname = $pim->getPatternInstanceName();
    $logesql = $piworkspace . "/" . $piname . "_Example_Flows/mqsi/Log.esql";
    unlink($logesql);
}
```

- Running Java code from PHP by using a static Java method. The following example shows a Java class, *MyClass*, that can be run from within a PHP script. The class contains a static method:

```
package com.your.company.domain.code;

import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyClass implements GeneratePatternInstanceTransform {

    public static void doSomethingUseful(String message) {
        System.out.println("Message received [" + message + "]");
    }

    @Override
    public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) { }
}
```

You can run the *MyClass* class from PHP as shown in the following example. In the line `$class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code.MyClass")`, the first argument, `com.your.company.domain.code`, is the ID of the plug-in that contains the Java class. The second argument, `com.your.company.domain.code.MyClass`, is the name of the class.

```
<?php
    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
    $class = $pim->getPluginClass("com.your.company.domain.code",
 "com.your.company.domain.code.MyClass");
    java_import("com.your.company.domain.code.MyClass");
    MyClass::doSomethingUseful("Hello!");
?>
```

- Running Java code from PHP by using a non-static Java method. The following example shows a Java class, *MyClass*, that can be run from within a PHP script. The class contains a method that is not static:

```
package com.your.company.domain.code;

import com.ibm.broker.config.appdev.MessageFlow;
import com.ibm.broker.config.appdev.patterns.GeneratePatternInstanceTransform;
import com.ibm.broker.config.appdev.patterns.PatternInstanceManager;

public class MyClass implements GeneratePatternInstanceTransform {

   public void doSomethingUseful(String message) {
      System.out.println("Message received [" + message + "]");
   }

   @Override
   public void onGeneratePatternInstance(PatternInstanceManager patternInstanceManager) { }
}
```

You can run the *MyClass* class from PHP as shown in the following example. In the line `$class = $pim->getPluginClass("com.your.company.domain.code", "com.your.company.domain.code.MyClass")`, the first argument, `com.your.company.domain.code`, is the ID of the plug-in that contains the Java class. The second argument, `com.your.company.domain.code.MyClass`, is the name of the class. A new instance of the class is created in the line `$obj = $class->newInstance()`. In the last line of the example, the `doSomethingUseful()` method of the class is run.

```
<?php
    $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
```

```
    $class = $pim->getPluginClass("com.your.company.domain.code",
 "com.your.company.domain.code.MyClass");
    $obj = $class->newInstance();
    $obj->doSomethingUseful("Hello!");
?>
```

- Using PHP scripts to read table values. In the following example, the PHP script extracts the values from an array stored in the superglobal variable *$_MB*. First the script searches inside the array for any values stored in the *event* and *response* variables, and then returns the values:

```
<?php
 $pim = $_MB["PATTERN_INSTANCE_MANAGER"];
 $table = $pim->getParameterTable("table1");
 $count = $table->getRowCount();
 for ($j=0;$j<$count ;$j++ ) {
  echo "".$_MB['PP']['table1'][$j]['event']."';\n";
  echo "".$_MB['PP']['table1'][$j]['response']."';\n";
 }
?>
```

  The array has the following structure:

```
array(4) {
 ["PATTERN_INSTANCE_MANAGER"]=>
 object(Java)#1 (0) {
 }
 ["PP"]=>
  array(10) {
   ["queueName"]=>string(11) "QP.QUEUE.QS"
   ["configurePrefixSuffix"]=>string(5) "false"
   ["queuePrefix"]=>string(3) "QP."
   ["queueSuffix"]=>string(3) ".QS"
   ["logging"]=>string(5) "queue"
   ["generateMessageSet"]=>string(4) "true"
   ["table1"]=>
   array(2) {
     [0]=>
      array(2) {
     ["event"]=>string(6) "event1"
     ["response"]=>string(9) "response1"
      }
     [1]=>
      array(2) {
     ["event"]=>string(6) "event2"
     ["response"]=>string(9) "response2"
      }
   }
 }
```

Testing PHP API code
After writing PHP code to modify pattern instances, test the code to check that it works correctly.

Before you begin

Before you test your PHP code, you must build the pattern plug-ins; see “Building pattern plug-ins” on page 2126.

About this task

You can check that your PHP scripts work correctly by creating an instance of your user-defined pattern. When you test your user-defined pattern, a new workspace is opened in which to create the instance of your pattern. If you want to change your PHP code and retest it, you are not required to open a new workspace after each change. If you make any other changes to the pattern authoring project, you must rebuild the pattern plug-ins and launch a new workspace.

Procedure

To test your user-defined pattern, see “Testing a user-defined pattern” on page 2126. A new workspace is opened in which you generate an instance of your user-defined pattern. After the pattern instance is

generated, check that your PHP code has worked correctly. For example, if you write PHP code to remove files from a pattern instance project, check that the files are deleted.

What to do next

You can either change your PHP code and retest it or you can package and distribute your user-defined pattern:

- To change and retest your PHP code, repeat the following steps until all your changes are complete. You can keep the test workspace open and continue to change your code and generate pattern instances without closing and relaunching the test workspace after each change:
  1. Edit the PHP code file in your main workspace and save the file.
  2. In the test workspace, generate a new instance of your user-defined pattern and check the results.
- To package and distribute your user-defined pattern, see "Packaging and distributing pattern plug-ins" on page 2127.

Building pattern plug-ins

You must build the pattern plug-ins before you can distribute user-defined patterns to pattern users.

Before you begin

Complete the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070

About this task

Procedure

To build the pattern plug-ins:

1. In the Pattern Authoring editor, click the **Create Pattern** tab.
2. Enter values in the **Plug-in ID** and **Version** fields for your plug-in according to the naming and version numbering conventions within your organization.

   For more information about naming conventions for Eclipse plug-ins, see http://wiki.eclipse.org/Naming_Conventions. For more information about version numbering conventions for Eclipse plug-ins, see http://wiki.eclipse.org/Version_Numbering.
3. Optional: Select **Create translation plug-ins**, if required.
4. Click **Create Pattern Plug-ins**.

   The plug-ins for your user-defined pattern are created and displayed under the Pattern Authoring folder in the Application Development view. The plug-ins are named *PluginID*, and *PluginID*.doc, where *PluginID* is the value entered in the **Plug-in ID** field in the **Create Pattern** tab. If **Create translation plug-ins** is selected, two additional plug-ins are created, named *PluginID*.nl1 and *PluginID*.nl1.doc.

What to do next

You must now test the pattern, see "Testing a user-defined pattern" on page 2126.

Testing a user-defined pattern

Ensure that you test your user-defined pattern before you share it with your pattern users.

Before you begin

Complete the following tasks:

- “Creating a pattern authoring project” on page 2069
- “Selecting the source files to use for a user-defined pattern” on page 2070
- “Building pattern plug-ins” on page 2126

About this task

Before you distribute your user-defined pattern, you can test it to ensure that it works correctly. To test your user-defined pattern, the Pattern Authoring editor opens a temporary workspace in which you can generate an instance of your user-defined pattern. You can then check that your user-defined pattern works correctly. If your user-defined pattern contains Java pattern authoring API code, see “Testing Java code” on page 2120. If your user-defined pattern contains PHP pattern authoring API code, see “Testing PHP API code” on page 2125.

Procedure

To test your user-defined pattern:

1. In the Pattern Authoring editor, click the **Create Pattern** tab.
2. Click **Test Pattern**.

   The **Workspace Launcher** window opens.
3. Select the temporary workspace in which to generate an instance of your user-defined pattern. Click **Browse**. Select a workspace or enter a workspace in the **Workspace** field. Click **OK**.

   The selected workspace opens.
4. Click the **Patterns Explorer** tab in the Application Development view.

   Your new user-defined pattern is shown in the **Patterns Explorer** view.
5. Test your pattern by generating a new pattern instance.

   See “Using patterns” on page 2058.
6. If you want to change your pattern, change the pattern authoring project and then rebuild the pattern plug-ins.

   If you are changing only Java or PHP pattern authoring API code, you do not have to rebuild the pattern plug-ins; see “Testing Java code” on page 2120 and “Testing PHP API code” on page 2125.

What to do next

After you have tested the pattern, you can share it with the pattern users, see “Packaging and distributing pattern plug-ins” on page 2127.

Packaging and distributing pattern plug-ins

Package the plug-ins for your user-defined pattern into a pattern archive so that pattern users can download and install the pattern on their own system.

Before you begin

Before you can complete this task, you must have completed the following tasks for your user-defined pattern:

1. “Creating a pattern authoring project” on page 2069
2. “Selecting the source files to use for a user-defined pattern” on page 2070
3. “Building pattern plug-ins” on page 2126
4. “Testing a user-defined pattern” on page 2126

About this task

To enable pattern users to use your user-defined pattern, you must create a pattern archive from the pattern plug-ins. The pattern archive is published to a pattern community site, which is either a website

or shared file system directory. The pattern user can download and install the pattern archive from the pattern community site. The following tasks describe this process:

- The pattern author creates a pattern archive, see "Creating a pattern archive" on page 2128.
- The pattern author uploads the pattern archive to a patterns community site, see "Uploading a pattern archive" on page 2129.
- The pattern user downloads and installs the pattern archive, see "Downloading and installing a pattern archive" on page 2130.

If you want to uninstall a user-defined pattern, see "Uninstalling a pattern archive" on page 2133.

Creating a pattern archive

Package your user-defined pattern as a pattern archive so that you can distribute it to pattern users.

Before you begin

Before you can complete this task, you must have completed the following tasks:

- "Creating a pattern authoring project" on page 2069
- "Selecting the source files to use for a user-defined pattern" on page 2070
- "Building pattern plug-ins" on page 2126
- "Testing a user-defined pattern" on page 2126

About this task

When you have tested your user-defined pattern and you are satisfied with its content and quality, package it into a pattern archive so that you can distribute it to your pattern users.

You can include additional plug-in projects in your pattern archive. For example, if your user-defined pattern includes a user-defined editor, you can include the code plug-in project that contains the code for the editor in your pattern archive. If you want to include a JAR archive package in your pattern archive, you must first convert the JAR file to a plug-in. To convert a JAR file to a plug-in, complete the following steps:

1. Click **File** > **New** > **Other**.
2. Expand **Plug-in Development** and select **Plug-in from Existing JAR Archives**.
3. Click **Next** to start the **New Plug-in from Existing JAR Archives** wizard. Navigate through the wizard, completing the fields as required. Click **Finish**.

If the code in the JAR plug-in is required by a code plug-in in your user-defined pattern, you must ensure that the `MANIFEST.MF` file of your code plug-in includes a dependency to the JAR plug-in.

Procedure

To package your user-defined pattern into a pattern archive, complete the following tasks:

1. Open your pattern instance project, click the **Create Pattern** tab.
2. In the Pattern Distribution section, click **Create Pattern Archive**.

   The **"Create a Pattern Archive"** window opens.

   The **File name** defaults to the first segment of the pattern project name appended by `.patternzip`
3. To select a location, click **Browse**, and navigate to the required location.

   The **Location** is the file system location to which the pattern archive is saved. If you want to publish the pattern archive directly to a pattern community shared file system location to which you have write access, select that location.
4. Select any additional plug-in projects from your workspace that you want to include in the pattern archive.

   Only plug-in projects are displayed in the list.

5. Click **Finish**.

   A progress monitor is displayed at the bottom of the window.

   - If you click **Cancel** while the creation process is being run, the progress monitor stops and a warning message is displayed saying that the operation has stopped and the pattern archive will not be created. You can choose to restart the creation process by clicking **Finish**, or cancel the creation process by clicking **Cancel**. If the pattern archive has already been created, it is removed from the target location and its resources are deleted from the workspace. If you click **Cancel** again, the window is closed.
   - If an error occurs during the creation process, for example, the disk is full, or a write access permissions problem has occurred, an error message is displayed. You can correct the problem without exiting the window, and try again.

   If the process completes successfully, the window closes.

Results

The pattern plug-ins and features are generated in the workspace. If the plug-ins already exist in the workspace, you are asked if you want to overwrite them; if you have already created the `.doc` plug-in or your translated files, you might choose not to overwrite the plug-ins. The pattern is temporarily displayed in the **Patterns Explorer** view. A pattern archive with the extension `.patternzip` is created in the target location.

What to do next

- If you published the pattern archive directly to a pattern community shared file system location, pattern users can now download the pattern archive, see "Downloading from a shared file system" on page 2132.
- If you did not publish the pattern archive directly to a pattern community shared file system location, you can upload the pattern archive to a pattern community site to distribute the pattern archive to pattern users, see "Uploading a pattern archive" on page 2129.

Uploading a pattern archive

The pattern author can upload a pattern archive to a pattern community site so that it can be distributed to pattern users.

Before you begin

Before you can complete this task, the pattern author must have completed the following task:

- "Creating a pattern archive" on page 2128

About this task

To publish a user-defined pattern, the pattern author must upload the pattern archive to a pattern community site, which is either a pattern community website or a pattern community shared file system location. To upload to a pattern community website, the pattern author must have write access to the website.

Procedure

- To upload to a pattern community website:
  - If the pattern community website supports the upload interface, the pattern author can specify the pattern metadata, for example, category, icon, and description, and then upload the pattern archive to the pattern community website.
  - If the pattern community website does not support the upload interface, the pattern archive must be passed to the administrator of the pattern community website so that it can be published.

- To upload to a pattern community shared file system location:
  - If the pattern author has write access to the shared file location, that location can be selected when the pattern archive is created. See “Creating a pattern archive” on page 2128.
  - If the pattern author does not have write access to the shared file location, the pattern archive must be passed to the administrator of the shared file location so that it can be published.

What to do next

The pattern user can now download and install the pattern archive, see “Downloading and installing a pattern archive” on page 2130.

Downloading and installing a pattern archive

To use a user-defined pattern, download and install the pattern archive for the user-defined pattern.

Before you begin

Before you can complete this task, you must decide which user-defined pattern you want to use from a pattern community site.

About this task

Before you can update a pattern archive to a newer version of an installed pattern archive, you must uninstall the current version of the pattern archive, see “Uninstalling a pattern archive” on page 2133.

When you have chosen a user-defined pattern that you want to use from a pattern community site, download and install the pattern archive for that user-defined pattern to your local system by using one of the following methods:

- “Downloading from a pattern community website” on page 2130
- “Downloading from a pattern community website by using a helper application” on page 2131
- “Downloading from a shared file system” on page 2132

What to do next

You can now use the user-defined pattern in your projects, see “Using patterns” on page 2058.

Downloading from a pattern community website

To use a user-defined pattern in your IBM App Connect Enterprise projects, download and install the pattern archive for the user-defined pattern from a pattern community website.

Before you begin

Before you can complete this task, you must have the URL of the user-defined pattern that you want to download.

In this installation use case, only HTTP URLs that have no security are supported.

About this task

If you want to stop the download process, click **Cancel**, the pattern is not installed and the tasks are rolled back. If you click **Cancel** again, the **Download Pattern** window closes.

Procedure

1. In the **Patterns Explorer** view, click **Download**.
   The **Download and Install Pattern** window opens.

2. Type in, or copy and paste, the complete URL for the pattern archive into the **File Location** field, click **Download**.

   The progress monitor starts to run and the status line shows the individual tasks that are being performed.

Results

The new user-defined pattern is displayed in the **Patterns Explorer** view.

What to do next

You can now use the user-defined pattern in your projects. See “Using patterns” on page 2058.

Downloading from a pattern community website by using a helper application
To use a user-defined pattern in your IBM App Connect Enterprise projects, download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application.

Before you begin

Before you can complete this task, you must be on the pattern community website that contains the pattern archive that you want to download.

About this task

You can install a pattern archive directly from a pattern community website by using a helper application.

Procedure

1. Navigate to the pattern archive that you want to download.
2. Click the link for the required pattern archive.

   A new window opens. The content of the window depends on your browser, but the typical options are to either open the file or save the file.
3. Install the pattern archive:

   - Windows On Windows:

     Select the option to open the pattern archive. The **Install Pattern Archive** window opens and shows a progress monitor.

     If you have more than one IBM App Connect Enterprise Toolkit installed on your workstation, the helper application is for the IBM App Connect Enterprise Toolkit that you most recently installed. If you want to control which IBM App Connect Enterprise Toolkit is extended with your pattern, download the file to disk, follow the instructions in “Downloading from a shared file system” on page 2132, then start the IBM App Connect Enterprise Toolkit that you want to extend.

   - Linux On Linux:

     The exact steps you must complete to install the pattern archive depend on your Linux distribution and the file explorer and browser you use. Use the following steps as a guide.

     a. Save the pattern archive to your local file system. Select the option in your browser window to save the pattern archive to your local file system. Choose a location for the file and start the download.

     b. After the download completes, open the pattern archive with the **`MBPatternInstaller`** helper application. Open a file explorer and navigate to the pattern archive. Use the **Open with** menu option on the pattern archive and select `MBPatternInstaller.bin`. The MBPatternInstaller

application is located in the IBM App Connect Enterprise Toolkit installation directory, for example:

- On Windows: `C:\Program Files\IBM\ACE\12.0.n.0\tools`
- On Linux: *`install_dir`*`/ace-12.0.n.0/tools` where *install_dir* is the directory into which the installation file was uncompressed.

. The **Install Pattern Archive** window opens and shows a progress monitor.

4. Close and relaunch the IBM App Connect Enterprise Toolkit.

Results

The new user-defined pattern is displayed in the **Patterns Explorer** view.

What to do next

You can now use the user-defined pattern in your projects. See "Using patterns" on page 2058.

Downloading from a shared file system
To use a user-defined pattern in your IBM App Connect Enterprise projects, download and install the pattern archive for the user-defined pattern from a pattern community site that is a shared file system by using the file URL or by browsing for a file.

Before you begin

Before you can complete this task, you must have access to the pattern community site that contains the pattern archive that you want to download.

About this task

- If you want to download and install the pattern archive for the user-defined pattern from a pattern community website by using a helper application and you have more than one IBM App Connect Enterprise Toolkit installed on your workstation, the helper application is for the IBM App Connect Enterprise Toolkit that you most recently installed. If you want to control which IBM App Connect Enterprise Toolkit is extended with your pattern, download the file to disk, complete the following instructions, then start the IBM App Connect Enterprise Toolkit that you want to extend.
- If you want to stop the download process, click **Cancel**, the pattern is not installed and the tasks are rolled back. If you click **Cancel** again, the **Download and Install Pattern** window closes.

Procedure

In the **Patterns Explorer** view, click **Download**.

The **Download and Install Pattern** window opens.

- If you want to download the pattern by using the file URL, type in, or copy and paste, the complete URL for the pattern archive into the **File Location** field, click **Download**. The progress monitor starts to run and the status line shows the individual tasks that are being performed.
- If you want to download the pattern by browsing your local file system:
  a. Click **Browse**. The **Select The Pattern Archive To Install** window opens.
  b. Navigate to the pattern archive that you want to use, click **Open**.
  c. Click **Download**. The progress monitor starts to run and the status line shows the individual tasks that are being performed.

Results

The new user-defined pattern is displayed in the **Patterns Explorer** view.

What to do next

You can now use the user-defined pattern in your projects; see “Using patterns” on page 2058.

Uninstalling a pattern archive

Uninstall the pattern archive to remove a user-defined pattern from the Patterns Explorer view.

Before you begin

Before you can complete this task, you must have completed the following task:

- “Downloading and installing a pattern archive” on page 2130

Each user-defined pattern archive has one pattern feature, the name of which is based on the name of the main pattern plug-in. If the name of the plug-in is *com.your.company.Test*, the name of the pattern feature is *com.your.company.Test.Feature_<plug-in version>*. If translation was selected by the pattern author, the pattern archive has a second pattern feature, *com.your.company.Test.Feature_NL.<plug-in version>*. To uninstall a pattern archive you must know the feature name of the pattern archive to ensure that the correct user-defined pattern is removed.

About this task

You can uninstall a pattern archive for a user-defined pattern that you have previously installed. Uninstalling a pattern archive removes the associated user-defined pattern from the Patterns Explorer view.

Procedure

To uninstall the pattern archive:

1. Click **Help** > **About**.
2. Click the **Installation Details** button.
3. Click the **Installed Software** tab.
4. Browse to and select the pattern feature. If translation was selected by the pattern author, two pattern features exist and both must be selected.
5. Click **Uninstall**.

   The **"Uninstall"** window opens.
6. Select the pattern feature and click **Finish**. The pattern feature is uninstalled.
7. You can either restart the workspace, or click **Apply Changes**.
8. Click **OK** to close the window.

Results

The pattern archive is removed and the user-defined pattern is removed from the Patterns Explorer view.

Constructing message models

You can model a wide variety of message formats by using XML Schema files, DFDL schema files, message sets, and WebSphere Adapters.

About this task

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134.

This topic area describes the concepts behind message modeling and the tasks that are involved in working with message models. If you are unfamiliar with message models, read the topics that describe

the concepts, starting with “Message modeling overview” on page 2134. These topics explain when you might want to model messages, and describe the message modeling objects that you can use, such as message model schema files, message sets, and message definition files.

The following tasks are provided for developing message models:

- “Modeling different data formats” on page 2207
- “Working with DFDL schema files” on page 2220
- “Working with XML schema files” on page 2242
- “Message Sets: Working with message sets” on page 2247
- “Applying a Quick Fix to a task list error” on page 2318
- “Importing files from the file system into the IBM App Connect Enterprise Toolkit” on page 2319

Message modeling overview

Much of the business world relies on the exchange of information between applications. This information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Applications typically use a combination of message formats, including those message formats that are defined by the following structures or standards:

- Comma Separated Values (CSV)
- COBOL, C, PL/I, and other language data structures
- Industry standards such as SWIFT, X12 or HL7
- XML including SOAP

You can model a wide variety of message formats so that they can be understood by IBM App Connect Enterprise message flows.

When the message format is known, the integration node can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow. After the message has been processed by the message flow, the integration node converts the message tree back into a message bit stream. Conversion to and from a bit stream is performed by a parser. IBM App Connect Enterprise supplies several parsers, each suited to a particular class of message formats called a domain.

The following topics together give an overview of Message modeling:

- “Message modeling concepts” on page 2134
- “Why model messages?” on page 2138
- “Message domains and parsers” on page 2146
- “The message model” on page 2147

Message modeling concepts

Message modeling is a way of predefining the message formats that are used by your applications.

Modeling your message formats is necessary for IBM App Connect Enterprise to understand some data formats, but simplifies the development of message flows regardless of the data format that you are processing. For a full description of the advantages of using message models, see “Why model messages?” on page 2138

When you model messages, you must understand the following concepts:

- Message models
- Schema files
- Domains and parsers
- New Message Model wizard

- Model editors
- Model generators
- Model validator

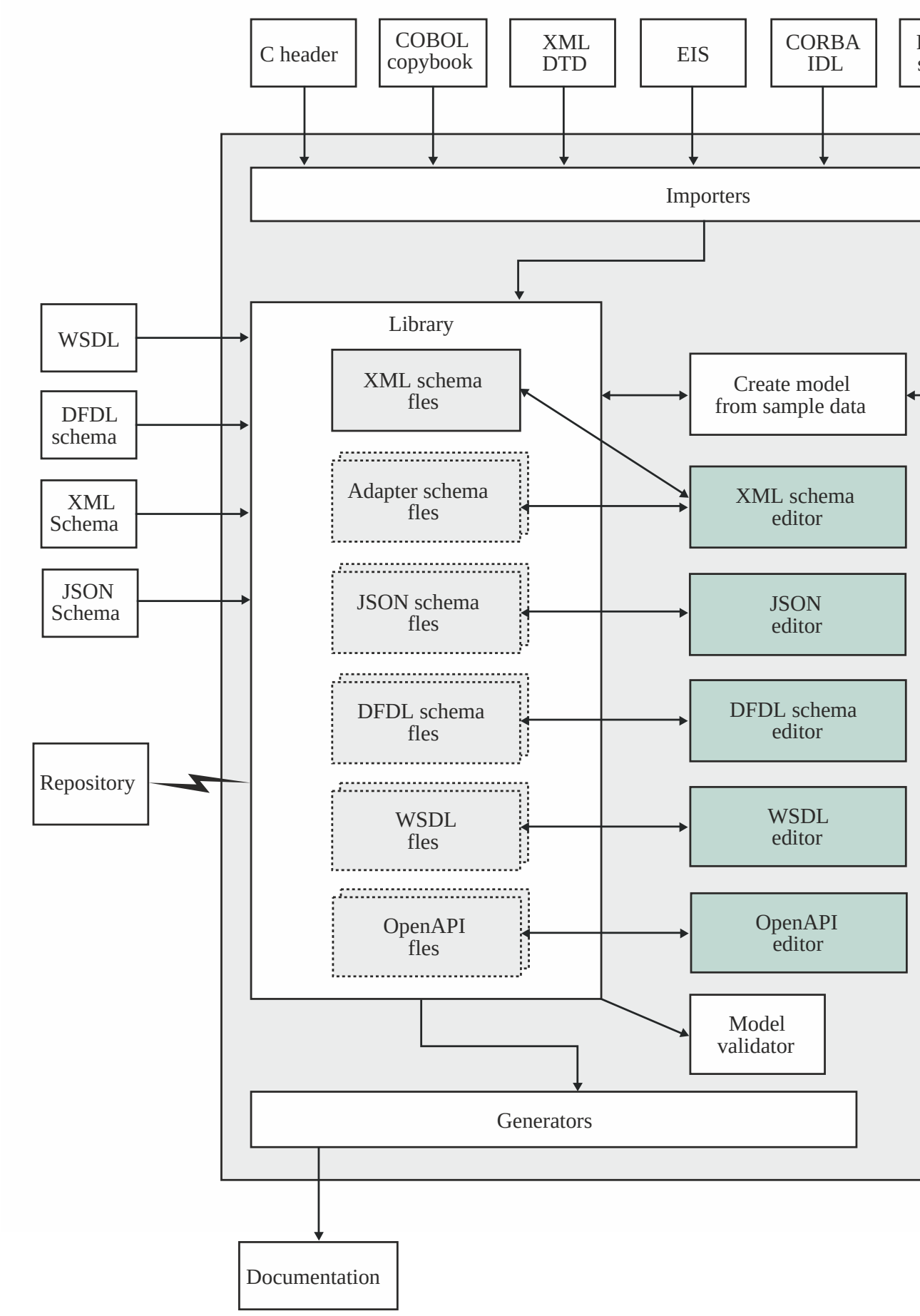
Additionally, you might have to understand the following concept:

- Message sets

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see “Enabling message set development” on page 2248 .

For more information about applications and libraries, see “Resource management overview” on page 1942. For a description of message sets, see “Message Sets: Message sets overview” on page 2169. The remainder of this topic describes message model schema files.

The following figure shows the different types of files that comprise a message model in a library or application. It shows the formats that can be imported by using the New Message Model wizard.


C header
COBOL copybook
XML DTD
EIS
CORBA IDL
Importers
Library
WSDL
DFDL schema
XML Schema
JSON Schema
Repository
XML schema fles
Adapter schema fles
JSON schema fles
DFDL schema fles
WSDL fles
OpenAPI fles
Create model from sample data
XML schema editor
JSON editor
DFDL schema editor
WSDL editor
OpenAPI editor
Model validator
Generators
Documentation

A message model is used by IBM App Connect Enterprise to model a message format. The message models used by IBM App Connect Enterprise are based on W3C XML Schema or the JSON Schema.

The following table shows the different message formats supported by IBM App Connect Enterprise, and the message model schema file that is used to model each format:

Format	**Message model schema file**
SOAP XML	XML Schema 1.0 and WSDL 1.0
Other XML	XML Schema 1.0
CSV	Data Format Description Language 1.0
Other delimited	Data Format Description Language 1.0
Industry standard text or binary data	Data Format Description Language 1.0
Custom text or binary data	Data Format Description Language 1.0
COBOL structure	Data Format Description Language 1.0
C structure	Data Format Description Language 1.0
PL/I structure	Data Format Description Language 1.0
CORBA IDL	XML Schema 1.0
SAP	WebSphere Adapter Schema or message sets
Siebel	WebSphere Adapter Schema
PeopleSoft	WebSphere Adapter Schema
JDEdwards	WebSphere Adapter Schema
Database record	XML Schema 1.0
JSON	JSON Schema
MIME	The MIME parser does not use a model.

XML Schema 1.0 (XSD) is an open standard modeling language from the World Wide Web Consortium (W3C) that was designed to model and validate XML documents. However, it can also be used to express the logical structure of many data formats. For more information about XML Schema, see “XML Schema” on page 2148.

Data Format Description Language 1.0 (DFDL) is an open standard modeling language from the Open Grid Forum (OGF) that builds upon the features of XSD 1.0 in order to model and validate all kinds of general text and binary data. It uses standard XSD model objects to express the logical structure of the data, together with DFDL annotations to describe the text or binary physical representation. For more information about DFDL, see “Data Format Description Language (DFDL)” on page 2162.

WebSphere Adapter Schema is an IBM extension to XSD 1.0. It uses the standard XSD model objects to express the logical structure of data, along with special annotations that are used when exchanging data with EIS systems that use the WebSphere Adapters of the integration node.

JSON Schema is a vocabulary that is designed to model and validate JSON documents. For more information about JSON Schema, see json-schema.org. There are several drafts of the JSON Schema specification; currently supported drafts are draft 4 and draft 5.

When you have created an IBM App Connect Enterprise application or library, you can add existing DFDL schema, XSD schema (and any associated WSDL files), JSON schema (and any associated OpenAPI definition files), and WebSphere Adapter schema files directly into your application or library, or add them to a folder within your project. Alternatively, you can create message model schema files by any of the following methods:

- o Using the XSD, DFDL, or JSON editors in the IBM App Connect Enterprise Toolkit

- o Importing from other files (for example XML DTD or COBOL copybooks)
- o Generating a model from an example data stream (XML only)
- o Connecting direct to EIS systems or databases
- o Running a tutorial that includes a schema

When a message model schema has been created, it can be edited to refine the model. Specific editors are provided for XSD schema and DFDL schema. Adapter schemas are edited using the XSD schema editor. JSON schemas are edited using the JSON editor. WSDL and OpenAPI definition editors are also provided. For more information about the available editors and their functions, see Editors.

A message model schema consists of a number of different model objects. For more information about message model objects, see “Message model objects” on page 2148.

When your message model schemas are complete, you can use them in developing your message flow. For more information, see “Why model messages?” on page 2138

Why model messages?

IBM App Connect Enterprise supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. However, most message formats are not self-defining, and a parser must have access to a predefined model that describes the message, if it is to parse the message correctly.

An example of a self-defining message format is XML. In XML, the message itself contains metadata in addition to data values, and it is this metadata that enables an XML parser to understand an XML message even if no model is available. Another example of a self-defining format is JSON.

Examples of messages that do not have a self-defining message format are CSV text messages, binary messages that originate from a COBOL program, and SWIFT formatted text messages. None of these message formats contain sufficient information to enable a parser to fully understand the message. In these cases, a model is required to describe them.

Even if your messages are self-defining, and do not require modeling, message modeling has the following advantages:

- Runtime validation of messages. Without a message model, a parser cannot check whether input and output messages have the correct structure and data values.
- Enhanced parsing of XML messages. Although XML is self-defining, all data values are treated as strings if a message model is not used. If a message model is used, the parser is provided with the data type of data values, and can cast the data accordingly.
- Improved productivity when writing ESQL. When you are creating ESQL programs for IBM App Connect Enterprise message flows, the ESQL editor can use message models to provide code completion assistance.
- Drag-and-drop operations on message maps. When you are creating message maps for IBM App Connect Enterprise message flows, the Graphical Data Mapping editor uses the message model to populate its source and target views.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.
- Generation of documentation.
- Provision of version control and access control for message models by storing them in a central repository.

To make full use of the facilities that are offered by IBM App Connect Enterprise, model your message formats.

To speed up the creation of message models, importers are provided to read metadata such as C header files, COBOL copybooks, and EIS metadata, and to create message models from that metadata. Additionally, predefined models are available for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG.

Ways to create message models

You can create a message model to model your data format.

You can create a message model by using the following methods:

- Importing an application message format that is described by an XML Schema, XML DTD, JSON Schema, C structure, COBOL structure, WSDL definition, or OpenAPI definition.
- By creating an empty message model file, then creating your message by using the editors provided in the IBM App Connect Enterprise Toolkit.
- By using the **Adapter Connection** wizard to import EIS metadata.
- By creating a populated model file from example message data.

Message model schema files

If your message models are to be stored as message model schema files, use the **New Message Model** wizard. If you already have XML schema files, JSON schema files, WSDL files, OpenAPI files, or DFDL schema files, you can import them directly into an application or library.

Message sets

If your message models are to be stored in message sets, use the **New Message Definition File** or **New Message Definition File From** wizards.

Additionally, you can import application message formats by using the **mqsicreatemsgdefs** or **mqsicreatemsgdefsfromwsdl** command-line utilities.

The **mqsicreatemsgdefs** command has a bulk import capability, but **mqsicreatemsgdefsfromwsdl** imports only one WSDL definition at a time.

Importing from other model representations to create message definitions

Importing other model representations is a convenient way to create an IBM App Connect Enterprise message model.

You can import the following message formats to create message sets:

- XML schema files
- XML DTD files
- JSON schema files
- C header files
- COBOL copybooks
- WSDL definitions
- EIS metadata
- Database records
- OpenAPI definitions

When you import one of these formats, a new message model is created to represent your message format. You can choose the name of the message model; if it already exists, the content is deleted and re-created as part of the import operation.

The new message model that is created can consist of both logical and physical information.

To find out which wizards to use to import message formats, see "Ways to create message models" on page 2139.

Tip: The following information is relevant only if you are using a message set to contain your message models.

You can also import C header files, COBOL copybooks, XML DTD files, or XML schema files by using the **mqsicreatemsgdefs** command-line utility. The **mqsicreatemsgdefs** command allows you to import

several middle-format files in a single operation, and allows you to create a message set (based on an existing message set) into which the message definition files are placed.

WSDL definitions can be imported by using the **mqsicreatemsgdefsfromwsdl** command-line utility. This utility imports only one WSDL definition at a time.

Message sets: Importing from XML Schemas to create message definitions

You can populate a message set with message definitions by importing XML Schema files, by using the **New Message Definition File From XML Schema file** wizard, the **Start from WSDL and/or XSD files** quick start wizard or the **mqsicreatemsgdefs** command.

Tip: This topic is relevant only if you are using a message set to contain your message models. If you are using an application or library, you can use one of the following methods to import XML schema files:

- Use the **New Message Model** wizard.
- Drop your XML schema files directly into your project; see "Creating an XML schema file by using the New Message Model wizard" on page 2244.

Each XML Schema file that you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML Schema file, but the **New Message Definition File From XML Schema file** wizard allows you to choose a different root file name.

If the message definition file already exists, you must have enabled overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

The namespace to which the message definition file created belongs depends on whether namespaces have been enabled for the message set.

- If namespaces have been enabled, the message definition file belongs to the target namespace of the XML Schema file that is imported.
- If namespaces have not been enabled for the message set, the message definition file belongs to the `noTarget` XML namespace irrespective of the target namespace of the imported XML Schema file and therefore is contained in the `(default)` location in your workspace.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

Import by using the New Message Definition File From XML Schema file wizard

When you import by using the **New Message Definition File From XML Schema file** wizard, you can specify which of the global elements or global complex types within the imported XML Schema file are to be messages within the message definition file.

You can import only one XML Schema file with each import operation. If your XML Schema file references other XML Schema files, with import or include elements, these XML Schema files must be imported into the same message set by using a separate import operation.

Import by using the command-line

When you import by using the command line you have the option of either creating no messages or creating a message for each global element and global complex type within the imported XML Schema file. The import operation creates a message and corresponding global element in the message definition file for each global element you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML Schema files in each import operation.

Physical information

In addition to creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, the physical format properties for *all* XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in

the new message set by default. If you want physical information created as well, see "Message Sets: Importing from the command line" on page 2301 for details.

MRM CWF and TDS physical format properties are *not* populated and so take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures you have imported are to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings, they do not prevent your model being generated in another form; for example, as a message dictionary.

Command-line invocation

The **mqsicreatemsgdefs** command-line utility allows you to import several XML Schema files in a single operation. All the XML Schema files must be in single directory, and the directory location must be passed as a parameter to the utility.

When you import into a message set for which namespaces are not enabled, the action to take for unsupported constructs can be specified by using an XML options file. This must contain an XML element called <XSD_NO_NS> that holds a set of information that applies to all XML Schema files that are imported during an invocation of the utility. A default XML options file, called `mqsicreatemsgdefs.xml`, is supplied. If you want to apply different sets of information to different XML Schema files, you must create multiple XML files and run the utility multiple times.

When you are importing into a message set for which namespaces are not enabled, there are two other options that you can specify in the <XSD_NO_NS> element in the XML options file:

- The prefix to use for the `http://www.w3.org/2001/XMLSchema-instance` namespace; the default is `xsi`.
- Additional namespace URI and prefix pairs.

The **mqsicreatemsgdefs** utility also allows you to create a message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains an XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is created by the import.

Further information about XML Schema

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Message sets: Importing from IBM supplied messages to create message definitions
You can add messages to a message set by using the **New Message Definition File From IBM supplied messages** wizard to import IBM supplied messages.

Tip: This topic is relevant only if you are using a message set to contain your message models. If you are using an application or library, use the **New Message Model** wizard, and select the data format with which you are working.

Each IBM supplied message that you import results in a new message definition file in the message set. The name of the message definition file defaults to the name of the IBM supplied message, but you can use the **New Message Definition File From IBM supplied messages** wizard to choose a different file name.

For information about what IBM supplied messages can be imported, see "Message Sets: Importing from the command line" on page 2301.

When you import message by using the **New Message Definition File From IBM supplied messages** wizard, you can specify only one IBM supplied message definition for each import operation.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

A report file is generated during the import operation that allows you to examine what occurred during the import process and check any errors that resulted.

Message sets: Importing from DTDs to create message definitions
You can populate a message set with message definitions by importing DTD files, by using either the **New Message Definition File From XML DTD file** wizard .

Tip: This topic is relevant only if you are using a message set to contain your message models. If you are using an application or library, use the **New Message Model** wizard, select **Other XML** as the data format that you are working with, then select **Create an XML schema by importing an XML DTD**.

Each XML DTD file that you import results in a new message definition file within the message set. The root name of the message definition file defaults to the root name of the XML DTD file, but the **New Message Definition File From XML DTD file** wizard allows you to choose a different root file name.

If the message definition file exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

All message definition files that are created as a result of DTD file import belong to the `noTarget` XML namespace and so are contained in the `(default)` location in your workspace.

A report file is created during the import operation, by default in the `log` folder of the message set. By default, it takes the name of the message definition file, with `.report.txt` appended.

Import by using the New Message Definition File From XML DTD file wizard

When you import by using the **"New Message Definition File From XML DTD file"** wizard, you can specify which of the elements within the imported XML DTD file are to be messages within the message definition file.

You can import only one XML DTD file with each import operation.

Import by using the command line

When you import by using the command line you have the option of either creating no messages or creating a message for each element within the imported XML DTD file. The import operation creates a message and a corresponding element in the message definition file for each element that you specify. If you do not specify that messages are to be created, you must create them manually using the message definition editor after the import has completed.

You can import several XML DTD files in each import operation.

Physical information

In addition to creating logical information, the import can also create physical information. If the message set contains any XML wire format physical formats, the physical format properties for *all* XML Wire Format layers is populated. If the message set does not contain any XML physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see “Message Sets: Importing from the command line” on page 2301 for details.

MRM CWF and TDS physical format properties are *not* populated and therefore take default values.

If you have one or more CWF or TDS layers, the import can cause entries in the task list, warning you that certain CWF or TDS properties must be set if the XML structures that you have imported are to appear in a CWF or TDS message.

If the CWF or TDS physical formats are not applicable to your XML structures, you can ignore these task list entries because they are just warnings; they do not prevent your model being generated in another form, such as a message dictionary.

Command-line invocation

The **mqsicreatemsgdefs** command-line utility allows you to import several XML DTD files in a single operation. All the XML DTD files must be in single directory, and the directory location must be passed as a parameter to the utility.

The **mqsicreatemsgdefs** utility also allows you to create a message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains an XML physical format and pre-populated message set level XML properties, which are then copied into the message set that is created by the import.

Further information about XML DTDs

For details about XML DTDs, see the World Wide Web Consortium (W3C) website.

Message sets: Importing from C header files to create message definitions
Create a message definition file from a C header file for use in the MRM and IDOC domains.

Tip: This topic is relevant only if you are using message definition files. As an alternative, consider the newer **New Message Model** wizard to create a schema for use in the DFDL domain; see Creating a DFDL schema file from a C header file.

You can populate your message set with message definitions by importing C header files, by using either the **New Message Definition File From C header file** wizard or the **mqsicreatemsgdefs** command.

Each C header file that you import results in a new message definition file. The root name of the message definition file defaults to the root name of the C header file, but the **"New Message Definition File From C header file"** wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

By default, all message definition files that are created as a result of an import from a C header file belong to the `noTarget` XML namespace and therefore reside in the `(default)` location in your workspace. This default namespace can be overridden by specifying a target namespace. See "Namespaces with non-XML messages" on page 2160 for reasons why you might want to do this.

In your C header file there are typically one or more C structures. You can select which of these structures to import. The import operation then imports those structures, plus any others that they require. All imported structures are converted into the equivalent elements, groups, and types in the message definition file.

You can also specify which of the selected structures are to be messages in the message definition file. The import operation creates a message and a corresponding global element in the message definition file for each structure that you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition editor after the import has completed.

If you import by using the **"New Message Definition File From C header file"** wizard you can import only one C header file with each import operation. But, if you import by using the command-line utility, you can import several C header files in each import operation.

If your C header file needs any other header files for a successful compilation, you must provide these and specify their location, because a compilation of your header file is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default, it takes the name of the message definition file, with `.report.txt` appended.

Physical information

In addition to creating logical information, the import can also create physical information.

If the message set contains any Custom Wire Format (CWF) physical formats, the physical format properties for all CWF layers are populated.

If the message set does not contain any CWF physical formats, only logical information is created. Also, if you import from the command line, only logical information is created in the new message set by default.

XML and TDS physical format properties are *not* populated and so take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the C structures you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your C structures, you can ignore these task list entries because they are just warnings; they do not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment-specific information as part of the import operation, and the necessary properties are set accordingly. There is a range of environments supported; if your environment is not shown, choose the closest match and review the created physical information by using the Message Definition Editor after the import has completed.

Command-line invocation

The **mqsicreatemsgdefs** command-line utility allows you to import several C header files in a single operation. All the C header files must be placed in the same directory and the directory location passed as a parameter to the utility.

You provide the necessary environment-specific information, and include file location information by using an XML file. This must contain an XML element called <C> which holds one set of information that applies to all C header files imported during an invocation of the utility. A default XML file called `mqsicreatemsgdefs.xml` is supplied. If you want to apply different sets of information to different header files, you must create multiple XML files and run the utility multiple times.

The **mqsicreatemsgdefs** utility also allows you to create message set into which the message definition files are placed, as part of the import operation. You can also choose to base this new message set on an existing message set. This facility enables you to prepare an empty message set containing a CWF physical format and message set level CWF properties already populated, which then gets copied into the message set created by the import.

Message sets: Importing from COBOL copybooks to create message definitions
You can populate your message set with message definitions by importing COBOL copybook files, by using the **"New Message Definition File From COBOL file"** wizard or the **mqsicreatemsgdefs** command.

Tip: This topic is relevant only if you are using a message set to contain your message models. If you are using an application or library, use the **New Message Model** wizard, select **COBOL** as the data format that you are working with, then select **Create a DFDL schema by importing a COBOL copybook or program**; see Creating a DFDL schema file by using the New Message Model wizard.

Each COBOL copybook that you import results in a new message definition file. The root name of the message definition file defaults to the root name of the COBOL copybook file, but the **"New Message Definition File From COBOL file"** wizard allows you to choose a different root file name.

If the message definition file already exists, you must have permitted overwriting to occur for the import to proceed, in which case the existing content is deleted and re-created.

By default, all message definition files that are created as a result of COBOL copybook file import belong to the `noTarget` XML namespace and therefore reside in the `(default)` location in your workspace. This default namespace can be overridden by specifying a target namespace. See "Namespaces with non-XML messages" on page 2160 for reasons why you might want to do this.

In your COBOL copybook file there are typically one or more level 01 structures. You can select which of these structures to import. The import operation then imports those structures, plus any others that they require. All imported structures are converted into the equivalent elements, groups, and types in the message definition file.

You can also specify which of the selected level 01 structures are to be messages in the message definition file. The import operation creates a message and corresponding global element in the message definition file for each structure that you specify. If you do not specify that messages are to be created, you must create them manually using the Message Definition Editor after the import has completed.

If you import by using the **"New Message Definition File From COBOL file"** wizard, you can import only one COBOL copybook file with each import operation. If you use the command-line utility, you can import several COBOL copybook files in each import operation.

If your COBOL copybook file needs any other copybooks in order to compile successfully, you must provide these in the same directory, because a compilation of your copybook is performed as part of the import operation.

A report file is created during the import operation. This is located by default in the `log` folder of the message set. By default it takes the name of the message definition file, with `.report.txt` appended.

The copybook must not contain field names that are COBOL reserved keywords.

Physical information

In addition to creating logical information, the import can also create physical information. If the message set contains any Custom Wire Format (CWF) physical formats, the physical format properties for all CWF layers are populated. If the message set does not contain any CWF physical formats, only logical information is created. If you import from the command line, only logical information is created in the new message set by default. If you want physical information created as well, see "Message Sets: Importing from the command line" on page 2301 for details.

XML and TDS physical format properties are *not* populated and therefore take default values.

If you have one or more TDS layers, the import can cause entries in the task list, warning you that certain TDS properties must be set if the COBOL structures that you have imported were to appear in a TDS message.

If the TDS physical format is not applicable to your COBOL structures, you can ignore these task list entries because they are just warnings, they do not prevent your model being generated in another form, such as a message dictionary.

Because physical information is created, the application target environment (platform and compiler) is important because it governs the way that, for example, integers appear in the message. You can specify environment-specific information as part of the import operation, and the necessary properties are set accordingly. There is a range of environments supported; if your environment is not shown, choose the closest match and review the created physical information by using the Message Definition Editor after the import has completed.

Command-line invocation

The **`mqsicreatemsgdefs`** command-line utility allows you to import several COBOL files in a single operation. All the COBOL copybook files must be in single directory, and the directory location passed as a parameter to the utility.

You provide the necessary environment-specific information by using an XML file. This must contain an XML element called <COBOL> that holds one set of environment-specific information that applies to all COBOL copybook files that are imported during an invocation of the utility. A default XML file called `mqsicreatemsgdefs.xml` is supplied. If you want to apply different sets of information to different copybooks, you must create multiple XML files and run the utility multiple times.

The **`mqsicreatemsgdefs`** utility also allows you to create message set into which the message definition files are placed, as part of the import operation. You can also choose to base the message set created on an existing message set. This facility enables you to prepare an empty message set that contains a CWF physical format and pre-populated message set level CWF properties, which are then copied into the message set that is created by the import.

Message sets: Importing WSDL files to create message definitions
Import WSDL files by using the **New Message Definition File From WSDL file** wizard, the **Start from WSDL and/or XSD files** Quick Start wizard, or the **mqsicreatemsgdefsfromwsdl** command.

Tip: This topic is relevant only if you are using a message set to contain your message models. If you are using an application or library, you can import your WSDL files, and any referenced XML schema files, directly into the application or library.

Each WSDL file that you import results in one or more new message definition files within the message set. A new message definition file is created for each namespace that is defined for the message set. The name of the message definition file defaults to the name of the WSDL file, but the **New Message Definition File From WSDL file** wizard allows you to choose a different file name.

If the message definition file exists, you must have permitted overwriting to occur for the import to proceed. The existing content is deleted and re-created.

The message set that you are importing the WSDL file into must be namespace-enabled. If it uses the MRM domain, it must have an XML physical format so that the message set is suitable for the runtime parsing of XML messages such as SOAP.

Use the report generated during the import operation to see what happened and to check any errors.

You specify a single WSDL definition for each import operation. If the WSDL definition consists of a hierarchy of files, you must supply the name of the file that contains the WSDL service or binding definitions. The WSDL definition that is imported must contain one or more WSDL bindings for the import to proceed. You can then select multiple bindings from the WSDL definition to define messages to model in the message set.

Importing by using the New Message Definition File wizard

When you import by using the New Message Definition File wizard, you can specify only one WSDL definition for each import operation. A WSDL definition can be held as one or more WSDL files, which are all imported as a result of importing the definition. The WSDL definition that is imported must contain one or more WSDL bindings for the import to proceed.

Importing by using the command line

The WSDL command-line importer (**mqsicreatemsgdefsfromwsdl**) can create a message set or update an existing one. If the message set project exists, it must be namespace-enabled and have an XML physical format layer. If the project does not exist, a new namespace-enabled project is created. If the import succeeds, new message definition files are added to the message set.

The **mqsicreatemsgdefsfromwsdl** command allows you to import one WSDL definition in a single operation.

The **mqsicreatemsgdefsfromwsdl** command copies the WSDL files it needs into the workspace before the import runs. These files are the top-level WSDL files and any imports are resolved by using an absolute or relative location. The files are copied under the specified message set in a folder called `importFiles`. They are not removed after the import; the user can later update or run validation on them in the IBM App Connect Enterprise Toolkit.

Physical information

An XML physical format layer is required for the MRM domain, and must be added to an existing message set before importing the WSDL definition.

Message domains and parsers

IBM App Connect Enterprise supplies a range of parsers to parse and write messages in different formats.

A parser is called when the bit stream that represents an input message must be converted to the format that is used internally by the integration node; this process is known as parsing. For more information

about the logical message tree structure created by parsing a message, see “Logical tree (message assembly)” on page 501.

A serializer is called when a logical tree structure must be converted into a bit stream; this process is known as serializing.

Each parser is suited to a particular class of messages, known as a message domain. The following list contains some examples of the message domains used in IBM App Connect Enterprise:

- XMLNSC - for XML documents
- DFDL - for general text or binary data streams including industry standards
- JSON - for JSON documents
- DataObject - for data without a stream representation

For more information, including the full list of parsers that are supplied with IBM App Connect Enterprise, see “Parsers” on page 520.

The message model

A message model is used by IBM App Connect Enterprise to model a message format. The message models used by IBM App Connect Enterprise are all based on either the World Wide Web Consortium (W3C) XML Schema 1.0 (XSD) or the JSON Schema, a standard from json-schema.org.

XML Schema is an international standard that defines a language for describing the structure of XML documents. It is suited to describing XML messages that flow between business applications, and it is widely used in the business community for this purpose. IBM App Connect Enterprise uses models that are based on XML Schema to describe the structure of XML messages, as well as messages in a number of other message formats.

JSON Schema defines a language for describing the structure of JSON documents. It is suited to describing JSON messages that flow between business applications, and it is widely used in the business community for this purpose. IBM App Connect Enterprise uses models that are based on JSON Schema to describe the structure of JSON messages. Draft 4 and draft 5 of JSON Schema are supported, including the extensions for OpenAPI V2.0 and V3.0.

Data Format Description Language 1.0 (DFDL) is an open standard modeling language from the Open Grid Forum (OGF) that builds upon the features of XML Schema 1.0 in order to model and validate all kinds of general text and binary data. It uses standard XSD model objects to describe the logical structure of data, together with DFDL annotations that describe the physical text or binary representation of data. IBM App Connect Enterprise uses DFDL schema files to describe text and binary data, including industry standard formats.

WebSphere Adapter Schema is an IBM extension to XML Schema 1.0. It uses standard XSD model objects to express the logical structure of data, together with IBM annotations that are used when exchanging data with EIS systems that use the WebSphere Adapters of the integration node.

Message definition files within message sets also use standard XSD model objects to express the logical structure of data, together with IBM annotations that describe the physical representation.

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see “Enabling message set development” on page 2248 .

To understand the different ways that you create and populate message model schema files, see “Ways to create message models” on page 2139.

XML Schema

XML Schema is an international standard from the World Wide Web Consortium (W3C) that defines a language for describing the structure of XML documents.

The XML Schema language is ideally suited to describing XML messages that flow between business applications, and it is widely used in the business community for this purpose.

IBM App Connect Enterprise uses XML Schema 1.0 to describe the logical structure of XML messages. At a simple level, the types and elements in the message are modeled by using XML Schema types and elements. However, when the need arises, all of the advanced modeling features of XML Schema are available for modeling messages.

IBM App Connect Enterprise defines some extensions of XML Schema, documented in "XML Schema extensions" on page 2148 and "Message Sets: XML Schema extensions in message sets" on page 2174. It also defines some restrictions, documented in "Message Sets: XML Schema restrictions in message sets" on page 2174.

Further information about XML Schema

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

XML Schema extensions
IBM App Connect Enterprise provides some additional facilities that are not specified in the XML Schema 1.0 specification.

Messages

A message is a global element that represents an entire message (rather than a structure within a message). Within an XML Schema file, a message is represented by a special global element that carries the extra information required by IBM App Connect Enterprise.

Further information about XML Schema

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Message model objects
An introduction to the objects that make up a message model. Message model objects are defined by XML Schema 1.0, except for Message which is an IBM App Connect Enterprise extension to XML Schema.

Message
A *message* describes the structure and content of a set of data that is exchanged between applications that send and receive the data. A *message* is a special complex element.

Simple element
A *simple element* describes one or more named data fields in a message. It is based on a *simple type* (for example, string, integer or float). A simple element can repeat, and it can define a default or a fixed value.

Simple type
A *simple type* describes a class of data within a message. It describes the type of data (for example, string, integer or float) and it can have value constraints which place limits on the values of any simple elements based on that simple type.

Complex element
A *complex element* describes a named complex structure within the message. The content of a complex element is defined by a *complex type*. A complex element can repeat.

Complex type
A *complex type* describes a complex structure within a message. It contains *elements* (simple or complex), *attributes* (if the data is XML), and *groups* that are organized into a tree-like hierarchy.

Group
A *group* describes a list of elements with information about how those elements can appear in a message. Groups can be ordered (sequence), unordered (all), or selective (choice). A group can repeat.

Attribute
An `attribute` describes an XML attribute. Attributes are similar to simple elements, but they require special treatment when used with XML messages. In messages that are not XML messages, attributes are not used.

Global and local objects

Most objects in the message model can be either global or local. A global object must have a unique name, which is used to refer to the object from one or more places in the message model. Local objects are defined and used in only one place in the message model.

Make objects local unless they must be used in more than one place. This reduces the probability of name clashes among the global objects in the message model, and makes the message set easier to work with.

Properties of message model objects

All message model objects have properties. The properties fall into three categories:

Logical
The logical properties of an object are defined by XML Schema. They relate to the format-independent description of the object called the 'logical model'. Logical properties describe *what* data the object contains without saying anything about *how* it is written down.

Physical
If the message model is for a data format that is not XML, additional physical properties are provided for an object that describe how the object is written down. These properties control the parsing and writing of the object. If the message model is a DFDL schema, then the properties are defined by the DFDL 1.0 specification. If the message model is in a message set, then the properties are an IBM proprietary set that is understood by the MRM domain and parser.

Documentation
This field is present for all message model objects. It provides a standard place for any description of the object that you might require. Text entered here does not affect the processing of messages in any way.

Message model objects: messages
A *message* describes the structure and content of a set of data that is passed from one application to another.

A message consists of elements that are organized into a logical structure agreed by the sending and receiving applications. This logical structure can be modeled so that message data can be parsed into a logical tree and manipulated easily by the integration node.

In the message model, a message is always based on a global element. The complex type of a global element describes the contents of the global element, and therefore describes all of the content of the message.

Multipart messages

If necessary, a message can contain other messages, and is necessary for modeling certain large and complex messaging standards such as SWIFT and EDIFACT. Such a message is known as a *multipart message*. The contained messages are known as *embedded messages*.

Message identification

Messages are identified by their name. In a message set only, a message can also be identified by an alias. The alias is an optional user-specified string that identifies the (multipart) message. The name and alias of a message must be unique within a message set.

Message model objects: elements

An *element* is a named piece of information (a field) within a message, with a meaning that is agreed by the applications that create and process the message.

An element has a specific meaning that is agreed by the applications that create and process the message. For example, a message might include a string that your applications have agreed is a 'Customer Name'. An element is always based on a type, either simple or complex.

An element:

- Has a business meaning.
- Is an instantiation of a simple or complex type.
- Can be accessed by name from ESQL, Java, or a Map.
- Is further defined by its type; for example, the type defines the range of values that an element can have.
- Can be defined globally or locally.

Simple and complex elements

Elements can be simple or complex. A simple element is a single, named piece of information such as 'Age' or 'Customer Name'. A simple element is based on a *simple type* that defines its content.

A complex element is a named structure that contains other elements. A complex element named 'Customer Details' might contain the simple elements 'Age' and 'Customer Name'. A complex element can also contain other complex elements. A complex element is based on a *complex type* that defines its content and structure.

Global and local elements

Elements can be global or local. A *global element* can be used in several different messages, or even in several places within the same message. It must be given a unique name by which it can be referenced by an *element reference*. A *local element* is defined in one position within one complex type or group, and is not available for reuse elsewhere in the message model.

Optional and repeating elements

Elements can be defined as optional, mandatory, and repeating, by using the properties `Min Occurs` and `Max Occurs`. For further information, see “Cardinality: optional, required, and repeating elements” on page 2157.

Default and fixed values

An element can be given a default value, so that if no value is supplied by the message, the default value is used. Alternatively, a fixed value can be defined, and the element always takes that value. The precise use of default and fixed values is dependent on the message domain.

Value constraints

The value of an element can be constrained by using *value constraints* which define the range of legal values for the element. The value constraints are associated with the simple type on which the element is based. For further information, see “Message model objects: simple types” on page 2152. The XML Schema term for a value constraint is a *facet*.

Defining substitution groups

If you are modeling XML messages, an element can be marked as a valid substitute for another element by using the substitution group property on the element. In this way, groups of elements can be assembled where any of the elements in the group can substitute for one element, the *head* element. For further information, see “Substitution groups in the message model” on page 2158.

Message model objects: types
Types describe the data content of elements.

Simple types describe simple elements with data types such as string, integer, or dateTime.

Complex types describe complex elements - elements that contain a hierarchy of other elements.

For more information, see:

- "Message model objects: simple types" on page 2152
- "Message model objects: complex types" on page 2151
- "Message model objects: type inheritance" on page 2153

Message model objects: complex types
A *complex type* describes the structure of one or more complex elements.

Complex types are an essential part of every message model because they define the logical structure of the messages and elements in the model.

What is a complex type for?

Complex types define the structure of the messages and elements in the message model. By combining elements, attributes, groups, and wildcards, almost any message structure can be modeled.

Contents of a complex type

Elements

Most complex types contain some elements, and some contain a large hierarchy of complex elements. The elements within a complex type are always contained within a group. This group can be local to the complex type, in which case the Message Definition Editor hides it from view.

Alternatively, the group that contains the elements can be a global group, and this group defines the element content, the composition, and the content validation for the complex type.

If a complex type is derived from a simple type, it cannot contain any element content.

Attributes
If you are modeling XML messages, your complex types can contain attributes. The attributes for a complex type can be local or global, and they can be contained within an attribute group.

Groups
Groups enable sets of elements to be included in a complex type. The members of the group are included as peers of the other elements. For more information about their use, see "Message model objects: groups" on page 2155.

Wildcards
If you are modeling XML messages, your complex types can contain wildcard elements. Wildcard elements enable unmodeled elements to be present in the complex type. Any such, elements must be present within the message at the same position as the wildcard. Complex types can also contain wildcard attributes. Wildcard attribute enable unmodeled attributes to be present within any elements that are based on the complex type.

Global and local complex types

Complex types can be global or local. A global complex type can be used as the basis for more than one complex element. It must be given a unique name by which it can be referenced. A local complex type is associated with a single complex element, and is not available for reuse elsewhere in the message model. Local types do not have a name, and are sometimes referred to as anonymous types.

Composition

The composition of a complex type describes how the members of the type are organized. For more information, see "Message model objects: groups" on page 2155.

Substitution settings

A complex type has parameters that control whether other types can be derived from it (final) and whether other types can substitute for it (block). For more information, see "Substitution groups in the message model" on page 2158 and "Message model objects: type inheritance" on page 2153.

Message model objects: simple types
A *simple type* is an abstract definition of an item of data such as a number, a string, or a date.

The purpose of a simple type is to define the content of one or more simple elements. Simple types (and any elements that are based on simple types) cannot contain attributes or child elements. Simple types stand in contrast to complex types, which define the structure of an element, but typically do not define any simple data.

Global and local simple types

Simple types can be global or local. A global simple type can be used as the basis for more than one element. It must be given a unique name by which it can be referenced. A local simple type is associated with a single element, and is not available for reuse elsewhere in the message model. Local types do not have a name and are sometimes referred to as anonymous types.

Variations of simple types

Built-in

XML Schema defines many simple types for you to use, covering all the standard data types such as strings, integers, decimals, and floats.

Restriction

You can define your own simple types by deriving from another simple type (the base type) by restriction. A restriction type can have value constraints applied to it.

A restriction type can derive from a built-in simple type or a restriction simple type.

List

For XML messages only, a list type is a way of rendering a repeating simple value in XML. The notation is more compact than the notation for a repeating element, and offers a way to have multi-valued attributes.

A list type can be based on a union type (introduced later in this section). This can describe a space-separated list of items in which each item can be based on any of the simple types in the union.

A list of lists is not legal. The item type of a list cannot be a list itself, or derived at any level from another list type.

A list type can have the facets of minLength, maxLength, and length applied to it. These facets restrict the number of items in the list. To restrict the values of each item in the list, facets must be applied to the item type and not to the list itself. The message definition editor provides additional support for enumeration and pattern facets directly on a List type, to enable the import of any schema that uses them, but issues a warning that enumeration and pattern facets are ignored by the integration node.

Union

A union type is a union of two or more other simple types.

A union type enables a value to conform to any one of several different simple types. The simple types that comprise a union type are known as its member types. There is no upper limit on how many member types can exist, but there must be at least one. A member type can be defined as a built-in simple type, a user defined simple type, or a local simple type defined anonymously within the union type.

A union type can also include list, union, and restricted simple types, among its members.

Value constraints

Value constraints are known as facets by XML Schema. Any value constraints that are applied to a derived type must further restrict the base type. It is not valid for a derived type to weaken or remove a value constraint that its base type has defined. If no value constraints are applied to the derived type, the derived type is almost identical to its base type, but it is treated as a restriction of the base type in situations where that is relevant (type inheritance and element substitution).

Message model objects: type inheritance
The XML Schema language allows a type definition to be based on another type definition. In this way, a hierarchy of types can be constructed.

This topic outlines the concepts of *type inheritance*, and highlights some important issues relating to substitution.

A full discussion of XML Schema type inheritance can be found on the World Wide Web Consortium (W3C) website, or in numerous books about XML Schema.

Restriction and extension

A type is a *restriction* of its base type, if elements of the derived type have a smaller range of valid values (or valid type members) than elements of the base type.

- For example, a restriction of a complex type might reduce the number of occurrences of one of its type members, or might omit that type member completely.
- Similarly, a restriction of a simple type might lower the `Max Inclusive` facet value, or raise the `Min Inclusive` facet value.

A type is an *extension* of its base type if elements of the derived type have a wider range of valid values (or valid type members) than elements of the base type.

- For example, an extension of a complex type might add type members that were not present in the base type, or might allow a type member to repeat.
- Similarly, an extension of a simple type must *always* be a complex type that is based on the simple type; you cannot extend a simple type by widening its range of valid values.

Special rules apply to the derivation of simple types. A simple type cannot extend another simple type. This ensures that restrictions that are imposed by a simple type cannot be removed by deriving another simple type from it.

However, a complex type can extend a simple type. This does not affect the range of valid values of the simple type, but it does allow attributes to be added. The result of extending a simple type is always a complex type that contains zero or more attributes.

Controlling type inheritance

The `final` attribute on a complex type can take three values, with the following effects:

- restriction: It is not valid to derive another complex type from this type by restriction.
- extension: It is not valid to derive another complex type from this type by extension.
- all: It is not valid to derive another complex type from this type by either extension or restriction

Type inheritance and substitution

XML Schema provides two different substitution mechanisms, both of which use type inheritance information to allow or disallow substitutions.

Element substitution is controlled by substitution groups, and element substitution can be blocked or allowed for extension and restriction by settings on either the element itself or the type of the element.

Type substitution allows the type of the element to be defined within the instance document, using the xsi:type attribute on the element, so that the real type of the element is not known until the element has been partly parsed. This mechanism can also be blocked or allowed based on the derivation method of the types involved.

Message model objects: simple type value constraints

Value constraints, also known as *facets* in XML Schema, refine a simple type by defining limits on the values that it can represent.

It is often useful to be able to constrain the values that an element or attribute can take, perhaps to ensure that messages conform to business rules. This topic describes how to add value constraints to a simple type, in order to constrain the values of all elements or attributes that are based on that simple type.

If the model is deployed to IBM App Connect Enterprise, elements and attributes can be validated against value constraints, so that violations are reported as errors or warnings. The DFDL, SOAP, XMLNSC, and MRM domains all support validation.

Types of value constraint

Length Constraints: `Length`, `Min Length`, `Max Length`

Using length constraints, the length of all elements based on the simple type can be constrained, or even limited to a single value.

Length constraints can be applied to simple types that are derived from xsd:hexBinary, xsd:base64Binary or xsd:string (including built-in schema types such as xsd:normalisedString).

Length constraints are inherited from ancestor types, and any length constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'longString' (`Max Length`=100) cannot be derived from a type 'shortString' (`Max Length`=10).

Range constraints: `Min Inclusive`, `Max Inclusive`, `Min Exclusive`, `Max Exclusive`

Range constraints specify the allowable range of values for all elements that are based on the simple type. Inclusive constraints include the specified endpoints in the permitted range, whereas exclusive constraints do not. Range constraints can be applied to simple types that are numeric, or that relate to calendar and time values. They cannot be applied to strings, because the ordering of string values depends on the character set that is used.

Range constraints are inherited from ancestor types, and any range constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'largeNumber' (`Max Inclusive`=100) cannot be derived from a type 'smallNumber' (`Max Inclusive`=10).

Enumeration constraints

An enumeration constraint specifies a single permitted value for all elements that are based on the simple type. A list of permitted values can be specified by defining more than one enumeration constraint for the same simple type. Enumeration constraints can be applied to all simple types.

Enumeration constraints are inherited from ancestor types, and any set of enumeration constraints that are defined for a simple type must not increase the range of permitted values. For example, a type 'AllColors' (with enumerations for all colors of the rainbow) cannot be derived from a type 'MonoColors' (with enumerations for 'black' and 'white' only).

Precision constraints: Total Digits and Fraction Digits

Precision constraints relate only to decimal and integer values. They limit the number of significant digits (total digits) and, for decimals, the number of decimal places (fraction digits) for all elements that are based on the simple type. Precision constraints can be applied to simple types that are derived from xsd:decimal and xsd:integer.

Precision constraints are inherited from ancestor types, and any precision constraints that are defined for a simple type must not relax the constraints that are imposed by any of its ancestor types. For example, a type 'veryPrecise' (`Fraction Digits`=10) cannot be derived from a type 'notVeryPrecise' (`Fraction Digits`=1).

Pattern constraints

A pattern constraint is a regular expression that specifies a set of permitted values for all elements that are based on the simple type. Multiple patterns can be defined for the same simple type, permitting complex validation rules to be expressed in logically separate parts. Each pattern

constraint on a simple type contributes to the set of permitted values for elements that are based on the simple type; that is, all the patterns are combined by using Boolean OR.

As with all value constraints, a simple type can inherit pattern constraints from the simple type on which it is based. In this case, the set of pattern constraints that are contributed by each ancestor type must be satisfied, in addition to the set that is contributed by the simple type itself; that is, the sets of pattern constraints from each level in the type hierarchy are combined by using Boolean AND.

White space constraints

A white space constraint specifies how a parser treats white space for all elements that are based on the simple type.

Note: For the DFDL and MRM domains, white space constraints are not applied.

Message model objects: groups

A *group* is a list of elements that defines how those elements appear in a message. Groups define the composition and content validation of a complex type.

In XML Schema, groups can be ordered (sequence), unordered (all), or selective (choice).

In DFDL Schema, groups can be ordered (sequence), unordered (sequence but with DFDL property dfdl:sequenceKind="unordered"), or selective (choice).

In message definition files in message sets, groups can be ordered (sequence), unordered (all), or selective (choice). For more information, see "Message Sets: XML Schema extensions in message sets" on page 2174.

What are groups for?

Groups can be used for any of the following purposes:

- To define the entire element content of a complex type.

  A complex type can refer to a global group that completely defines its content. (If it does not, the content of the complex type is defined by an *anonymous local group*, which is hidden within the Message Definition Editor.)
- To represent a common substructure within more than one type.

  Two or more complex types can refer to the same global group, if they both contain the same subset of elements.
- To change the composition midway through a complex type.

  You might have a complex type that is a sequence of three members, but the second member is a choice of two elements. To model this circumstance, a group with `composition` set to choice can be inserted as the second member of the sequence.

Contents of a group

Groups can contain complex elements, simple elements, wildcard elements, and groups.

By combining these components, the structure of any message can be modeled. Wildcard elements can be included to enable the presence of unmodeled elements, thus making the message model robust and flexible.

Global and local groups

Groups can be global or local.

A *global group* can be used in more than one place in the message model. It represents a structure that is present in more than one place in the message model. A global group must be given a unique name by which it can be referenced by a *group reference*.

A *local group* is defined in one position within one group, and is not available for reuse elsewhere in the message model. Local groups do not have a name, and are displayed by using the composition of the group.

Message model objects: attributes
An *attribute* describes an XML attribute. They are used only when the data is XML.

Global and local attributes

Attributes can be global or local.

A *global attribute* can be used in more than one place in the message model. It must be given a unique name by which it can be referenced by an *attribute reference*.

A *local attribute* is defined in one position within one complex type, and cannot be used elsewhere in the message model.

Optional attributes

Attributes can be defined in XML Schema as optional, required, or prohibited. Attributes cannot repeat. For further information, see "Cardinality: optional, required, and repeating elements" on page 2157.

Default and fixed values

An attribute can be given a default value so that, if the attribute is missing from the message, the default is used. Alternatively, a fixed value can be defined, and the attribute always takes that value. The precise use of default and fixed values is dependent on the message domain.

Value constraints

The value of an attribute can be constrained by using *value constraints*, which define the range of legal values for the attribute. Value constraints are associated with the simple type on which the attribute is based. For more details, see "Message model objects: simple types" on page 2152. In XML Schema, the term for *value constraint* is *facet*.

Message model objects: attribute groups
For XML messages, an *attribute group* defines a set of attributes that can be present in a complex type.

An attribute group provides a way to include the same set of attributes in more than one complex type, without duplicating the definitions.

For example, if most of the elements in your message model have the attributes 'amount', 'currency' and 'date', these elements can be contained in a single attribute group, which is referenced by all the complex types that use them.

Message model objects: Wildcard elements
For XML messages, a wildcard element represents an element that is not present in the message model, but which might be present at the position of the wildcard element in a message.

Wildcard elements provide a means of adding flexibility to the message model, so that messages can be parsed even if they do not exactly match the message model.

Wildcard elements can be present only within a complex type or group with `Composition` of sequence.

The `Process Content` and `Namespace` properties control the namespace to which elements that are present at the position of the wildcard element must belong.

Message model objects: Wildcard attributes
For XML messages, a *wildcard attribute* enables unmodeled attributes to be present in a message.

The `Process Content` and `Namespace` properties control the namespace to which attributes that are present in the position of the wildcard must belong.

Message model object identification
Objects in a message model (elements, attributes, types, groups) are identified by their name only.

This means that no two objects in the same scope can have the same name. Name clashes can be avoided more easily if global objects are used only when necessary. Local objects are not visible outside of the scope of their parent object, so their names can be reused without causing a name clash.

Namespaces
Each message model schema can specify a namespace. Namespaces are an XML Schema mechanism for organizing groups of related objects into a named 'module'.

Global objects in different namespaces can share the same name. Therefore, namespaces offer another means of avoiding name clashes among global objects.

Valid names
Since the message model is based on the XML Schema language, the name of every message model object must be a valid XML Schema identifier. For information about what constitutes a valid XML Schema identifier, see XML Schema Part 0: Primer.

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Cardinality: optional, required, and repeating elements
The number of occurrences of an element can be controlled using the properties `Min Occurs` and `Max Occurs`. Using these properties, an element can be defined as mandatory, optional or repeating.

Elements

A *mandatory*, or required, element has `Min Occurs`>= 1. A mandatory element must occur at least once in the message.

An *optional* element has `Min Occurs` = 0. An optional element can be omitted from the message.

A *repeating* element has `Max Occurs`> 1 to indicate a bounded number of repeats, or `Max Occurs`=unbounded (sometimes displayed as -1), to indicate an unlimited number of repeats. A repeating element occurs more than once in the message, and all the occurrences must appear together without any other elements between them.

If a complex type or a group contains two, or more, members that refer to the same element, the second reference is a *duplicate*. This is different from a repeating element, because the two references are typically separated by other members of the type or group. In the message, the second occurrence typically does not appear immediately after the first occurrence.

Attributes

The number of occurrences of an attribute can be controlled by setting it to *required*, *optional* or *prohibited*.

A *required* attribute is similar to a mandatory element - it must occur in the message.

An *optional* attribute is similar to an optional element - it can be omitted from the message.

A *prohibited* attribute must not appear in the message.

An attribute is not allowed to repeat, and duplicate attribute references are not allowed within an attribute group.

Self-defining elements and messages
An instance element is *predefined* if it is possible for the parser to find a matching element definition in the message model with an appropriate set of properties and in the correct context. Otherwise, it is *self-defining*. Similarly, an entire message is self-defining if no corresponding message is present in the message model.

Self-defining elements can be used only when the format of the message is a self-describing one, such as XML or JSON. For general text or binary formats (for example, comma separated), you must ensure that your message model defines every message and every element that must be parsed.

If you have chosen not to model your messages, or you have a model but have chosen not to deploy it to the integration node, all messages and elements are self-defining. In this situation, you cannot use message definitions to influence the parsing and writing of elements; the self-defining elements are parsed and written according to the default behavior of the parser and writer.

Self-defining elements, and all elements within a self-defining message, are not validated against value constraints, and any missing fields are not assigned default or fixed values, and all data is assumed to be string type unless the parser is able to deduce the type in a reliable manner.

Substitution groups in the message model
Substitution groups are an XML Schema feature that provides a way of substituting one element for another in an XML message.

A substitution group is a list of global elements that can be present in place of another global element, called the *head element*.

A substitution group is defined by setting the `substitution group` property on one global element (the *member* element) to point at another global element (the *head* element). This adds the member element to the substitution group of the head element.

Head elements
A head element is an element that can be substituted. When a message is parsed, one of its member elements can be present in place of the head element without causing a validation error.

Abstract elements
An abstract element is a head element which must be substituted, and is indicated by the 'abstract' attribute on the element. Typically, abstract elements have other elements in their substitution group, otherwise they are of little use. Wherever an abstract element is present in a message definition, a member of its substitution group must be present instead.

The `block` attribute on elements
The `block` attribute on an element limits the set of global elements that can substitute for the element. The `block` attribute can take any subset of the values restriction, extension, substitution, or all.

- If the `block` attribute contains restriction, an element that is based on a restriction of the type of the element cannot be substituted for the element.
- If the `block` attribute contains extension, an element that is based on an extension of the type of the element cannot be substituted for the element.
- If the `block` attribute contains substitution, an element that is a member of the substitution group of the element cannot be substituted for the element.
- If the `block` attribute contains all, all of the above limits apply.

The `final` attribute on elements
The `final` attribute on an element limits the set of global elements that can be a member of the substitution group of the element. The `final` attribute can take any subset of the values restriction, extension, or all.

- If the `final` attribute contains restriction, an element that is based on a restriction of the type of the element cannot be in the substitution group of the element.
- If the `final` attribute contains extension, an element that is based on an extension of the type of the element cannot be in the substitution group of the element.
- If the `final` attribute contains all, both of the above limits apply.

The `block` attribute on complex types
The `block` attribute on a complex type limits the set of other types that can substitute for that type. The `block` attribute can take values restriction, extension, or all. The meanings for these values are the same as the values that are shown for the block attribute on an element. An element that is a member of a substitution group can substitute only for the head element if its type is compatible with the block attribute on the type of the head element.

Default `block` and `final` attributes
A default for the `block` and `final` attributes can be set at the message definition file level. If a default for one or both of these attributes has been set and the relevant `block` or `final` attribute has not been set at the object level, the default setting is used for that object. You can override the default setting at the object level.

Namespaces in the message model
Use namespaces to qualify message model object names.

A single XML instance document can contain elements and attributes that are defined for, and possibly used by, multiple applications. Two different elements or attributes within the same document might require the same name. Individual applications must be able to recognize the elements and attributes that they are designed to process. In circumstances such as this, the definitions can be distinguished from each other by qualifying each element with a different namespace. This avoids problems of name collision and mistaken recognition.

XML Schemas can define a target namespace. Global elements, attributes, groups, and types that are defined within an XML Schema are qualified by the target namespace, if it has been defined. Optionally, local elements and attributes can also be qualified by the target namespace. Therefore, namespaces assist in the development of a library of XML Schemas that can be developed independently. If the namespace name that is used for an XML Schema is unique, you do not have to be concerned about name clashes with objects that are defined within other XML Schemas.

The scope of a namespace extends beyond the scope of its containing document and is identified by a Uniform Resource Identifier (URI). In order to serve its purpose, a URI must be unique. You might be more familiar with the concept of a Universal Resource Locator (URL). URIs often use the same syntax as URLs, but the URI definition is broader than the specification of a URL. This is an example of a URI: `http://mycompany.com/xml_schema`

A namespace prefix is declared as a shorthand for the full URI name and this is used to qualify all elements that belong to that namespace. The prefix to be substituted for a namespace in an XML instance document or XML Schema is specified by using an `xmlns` attribute. A default namespace can also be defined by using an `xmlns` attribute. If a default namespace is defined, any element or attribute with no prefix is qualified with the default namespace. If no default namespace is defined, any element or attribute with no prefix is not qualified by a namespace.

Namespaces and the message model

Message model schema files in applications or libraries, and message definition files in message sets, support namespaces.

A single application, library, or message set can contain a number of different namespaces. Each namespace is represented by a different message model schema file or message definition file. When you create a file, you can choose whether it has an associated namespace, or whether it is in the notarget namespace. If you associate a namespace, you must also choose a prefix.

If the file has an associated namespace, the following global objects are qualified with the namespace:

- Elements
- Attributes
- Simple Types
- Complex Types
- Groups
- Attribute Groups

Optionally, local elements and attributes can be qualified with the namespace.

Objects that are defined in a file can reference objects in other files. To create these references, import or include one file within another file. A message model schema file can import or include another file in the same application or library, or in a separate IBM App Connect Enterprise library. A message definition file can import or include another file only if it is in the same message set.

Message parsing and message flows

IBM App Connect Enterprise parsers for XML data recognize prefixed names in the XML messages that they parse, and internally map these to the correct namespace. The message tree stores the name and the namespace of the element or attribute.

Namespaces can be used even when the data is not XML. DFDL schema, Adapter schema, and message definition files can be created with an associated target namespace. Though the data itself does not contain prefixed names, the namespace is obtained from the corresponding element in the file. Again, the message tree stores the name and the namespace of the element.

You can specify namespaces when writing ESQL or Java. It is not necessary to write ESQL or Java that is namespace aware if you are not using namespaces. However, if you decide to use namespaces it is necessary to write namespace-aware ESQL or Java. The namespace in which an element is contained is stored in the message tree when parsed. This is a logical property and it is held regardless of the physical wire format in which messages are parsed and written. Syntax has been added to ESQL to make it easy to reference namespaces of other elements by using defined prefixes. In Java, XPath expressions are used to reference elements.

Further information about XML

On the World Wide Web Consortium (W3C) website, see:

- Extensible Markup Language (XML)
- XML Schema Part 0: Primer
- Namespaces in XML

Namespaces with non-XML messages
The use of namespaces by IBM App Connect Enterprise is not necessarily limited to XML message models.

There is one scenario where the use of namespaces by non-XML message models can simplify the ESQL or Java code that you write. Before describing this scenario, it is important to understand that the DFDL and MRM parsers, when parsing messages that are defined in a file that has a target namespace, produce a logical message tree that contains both name and namespace information. For non-XML messages, the namespace is obtained from the element declaration in the file, and not from the data.

Consider a transformation scenario where a message from a COBOL application requires to be transformed into namespace-aware XML; for example, a SOAP XML message. The transform must map the logical message tree that was created for the COBOL message to a logical message tree that matches the XML message. If the COBOL message tree does not contain namespace information, each mapping from a COBOL field to an XML element must set the namespace for the XML element. However, if the COBOL message tree already contains the required namespace information, this mapping is much simpler.

To enable the DFDL or MRM parser to create namespace information in a message tree that was created from a non-XML message, you must specify a target namespace for the DFDL schema or message definition file. For MRM, this must be done as part of the file creation process; for DFDL this must be done after the file has been created in the DFDL editor. Make the target namespace of the file the same as the target namespace of the XML message into which the non-XML message is being transformed.

When dealing with both the message tree for the non-XML message and the message tree for the XML message, the ESQL or Java code that you write to perform the transformation must be namespace aware.

Specifying namespaces in the Message property
When using the DFDL or MRM domains, the *Message* property (formerly the *Message Type* property) of a node is used to specify the name of the message.

The format of a simple message is `{namespace-uri}:name` where *name* is the name of the message, and *namespace-uri.* identifies the namespace. The namespace must be the full URI specification and must be enclosed in braces.

If you omit `{namespace-uri}`, the first match for the name that is found in the model is used. You should omit `{namespace-uri}` only if a name is unique within a message model.

For MRM only, the format `{namespace-uri}name` (that is, with no colon) is also valid. This maintains compatibility with previous version.

The following are examples of message property types:

- A simple message property for a message in a real target namespace: `{http://www.ibm.com/space}:myMessage`
- A simple message property for a message in the notarget namespace: `{}:myMessage`
- A simple message property for a message in any namespace: `myMessage`

Reusing message model files
One message model schema file or message definition file can reuse message model objects defined in another file.

There are two mechanisms that XML Schema provides to reuse message definition files: **`import`** and **`include`**. The namespaces of the two files determine which of the **`import`** or **`include`** commands is be used:

	Target file has a target namespace	**Target file has notarget namespace**
Parent file has a target namespace	xsd:import	xsd:include[1]
Parent file has notarget namespace	xsd:import	xsd:include

Note: When a target namespace file includes a notarget namespace file, referencing an object in the target file from the parent file causes the object to be present in the namespace of the parent file.

When **`import`** or **`include`** are used, global objects from the target file can be used in the parent file. For example, an element in the parent file can be given a complex type defined in the target file.

The namespace of objects in the target file is preserved in the parent file, with the exception noted in the previous table of a target namespace file that includes a notarget namespace file. This exception is sometimes called the chameleon namespace effect.

A message model schema file can import or include another file in the same application or library, or in a separate IBM App Connect Enterprise library. A message definition file can import or include another file only if it is in the same message set.

XML Schema provides a variation of xsd:include called xsd:redefine, which is supported when using message model schema files, but not when using message definition files. A Quick Fix is offered to convert occurrences of xsd:redefine into xsd:include when using message definition files.

Namespaces with MRM XML messages
The namespace that is associated with a message definition file is part of the logical layer of the message model.

Therefore, it is not dependent on an XML Wire Format being present. However, if you have an XML Wire Format, the namespace information from the logical layer is used to populate some of the properties of the XML Wire Format. If namespaces are enabled for a Message Set, in the XML Wire Format, a table of namespace URI/prefix pairs is maintained. This table is initially populated with the namespaces of all of the Message Definition Files with their prefixes when they were created.

If your message set has namespaces enabled, the integration node does not store the values of any `xmlns` attributes in the tree when it parses an XML instance document. It also does not store the values of any `Schema Location` and `No Namespace Schema Location` attributes. When an XML document is written out, the integration node regenerates this information from the properties of the XML Wire Format of the message set.

The table of namespace URI/prefix pairs is used by the MRM Domain when it produces an XML message. Elements and attributes that are qualified by a namespace are prefixed with the corresponding prefix from the table. The integration node also manages the output of the corresponding `xmlns` attributes that map the prefixes to namespaces. You can choose whether `xmlns` attributes for all of the entries in the namespace URI/prefix table are written at the start of the document, or whether they are only written in the document when required.

If namespaces are enabled for a Message Set, in the XML Wire Format there is a table of schema locations that map namespace URIs to file names. You can add entries to this table and you can map a file name to the notarget namespace. If you are using IBM App Connect Enterprise, this table is used to produce `schemaLocation` and `No Namespace Schema Location` attributes at the start of the XML document.

JSON Schema

JSON Schema is a standard from json-schema.org that defines a language for describing the structure of JSON documents.

The JSON Schema language is ideally suited to describing JSON messages that flow between business applications, and it is widely used in the business community for this purpose.

IBM App Connect Enterprise uses JSON Schema to describe the structure of JSON messages. Draft 4 and draft 5 of JSON Schema are supported, including the extensions for OpenAPI V2.0 and V3.0.

For details about JSON Schema, see json-schema.org

Data Format Description Language (DFDL)

Data Format Description Language (DFDL) 1.0 is a modeling language from the Open Grid Forum that is used to define the structure of general text and binary formatted data in a way that is independent of the data format. It is based on XML Schema 1.0.

DFDL is a way of describing the data. It is not a data format. DFDL can describe many data formats, including:

- Textual and binary
- Commercial record-oriented
- Scientific and numeric
- Modern and legacy
- Industry standards

DFDL schema files use XML Schema objects, and annotations on those objects to define the data.

- The XML Schema objects define the logical format of the data. You cannot use XML attributes in the data model.
- DFDL schema annotations describe the physical format of the data.
- XPath expressions are used to reference fields within the data.

DFDL is not intended to be used to model XML documents. Use normal XML Schema files to model XML.

For a simple example, see Example DFDL schema.

Support for DFDL

Support for DFDL in this product includes:

- DFDL parser and domain.
- DFDL parser (C and Java).
- DFDL serializer (C and Java).
- DFDL schema file creation wizards. See “Creating a DFDL schema file” on page 2223.
- DFDL schema editor for modeling text and binary data formats. See DFDL schema editor.
- DFDL Test perspective for testing your DFDL schema files. See “Testing a DFDL schema file” on page 2226.
- Developing independent applications that use the IBM DFDL API. See “Developing independent DFDL applications” on page 2239.

Not all features of the DFDL 1.0 specification are supported, and this implementation has some limitations. See the following topics:

- “Unsupported features” on page 2164
- “Implementation-specific limits” on page 2168

Further information about DFDL 1.0

For more information about DFDL, see the Open Grid Forum (OGF) website.

A navigable copy of the DFDL v1.0 Specification is included with this information. The specification is produced by the Open Grid Forum (OGF) DFDL Working Group and is an OGF Grid Recommendation. The master specification documents are hosted at https://github.com/OpenGridForum/DFDL.

The OGF document repository also contains a DFDL tutorial, comprising a number of PDF files. IBM have several DFDL modeling scenarios available on developerWorks.

DFDL extensions
Additional facilities available in this product that are not specified in the DFDL 1.0 specification.

Messages

A message is a global element that represents an entire data stream (rather than a structure within a larger data stream). Within a DFDL schema file, a message is represented by a special IBM annotation on a global element.

Further information about DFDL

For more information about DFDL, see the Open Grid Forum web site. This site has tutorials on DFDL.

DFDL encoding information
When an integration node calls a parser or serializer, the integration node provides the parser or serializer with encoding information that defines the character encoding (CCSID) and numeric encoding (byte order and floating point representation) of the data that is to be parsed or serialized.

When parsing or serializing messages in the DFDL domain, the DFDL parser or DFDL serializer is called. The way that the DFDL parser and DFDL serializer use encoding information is defined by your DFDL message model.

In a DFDL message model, you must set the character encoding (CCSID) and numeric encoding (byte order and floating point representation) by using DFDL properties on each type definition. DFDL predefines a set of external variables, each of which has default values that can be externally overridden. It is these predefined variables that the integration node overrides to pass in encoding information. For more information about DFDL predefined variables, see DFDL predefined variables in.

If you have set the encoding properties in your DFDL message model to specific, static types, the DFDL parser and DFDL serializer interpret message data by using this static encoding information, and ignore the encoding information that is provided by the integration node for each individual message instance. Note that if you set the encoding properties in your DFDL message model to specific, static types, and the message data is not encoded in the format that you specified, you might encounter the following problems:

- Parsing errors due to the message data being interpreted incorrectly
- Incorrect numeric values due to the wrong byte order (endianness) being used

To avoid these problems, set the DFDL encoding properties in your DFDL message model as follows:

- encoding: `{dfdl:encoding}`
- byteOrder: `{dfdl:byteOrder}`
- binaryFloatRep: `{dfdl:binaryFloatRep}`

These settings allow the DFDL parser to correctly interpret your data by using the encoding information that the integration node obtains for an individual message instance, for example from the transport headers in an input message. Similarly, these settings ensure that bit stream data is encoded according

to the values that are defined in the integration node properties, when the data is processed by the DFDL serializer.

DFDL schema version and keywords
When you develop a DFDL schema, you can define key information that you want to associate with it, including the file version.

After you have deployed your DFDL schema in a BAR file, you can view the properties of the DFDL schema file. The properties of the DFDL schema file include the deployment dates and times, the modification dates and times (the default information that is displayed), and any additional keyword information that you have set.

Keywords

You can define keywords anywhere in your DFDL schema by using either XML comments or the XML Schema documentation annotation facility. Keywords must follow certain rules to ensure that the information contained in the file can be parsed. The following examples show the type of information that you can define:

- `$MQSI Version=1.2 MQSI$`
- `$MQSI Author=John Smith MQSI$`

The following table contains the information that is displayed:

DFDL schema name	
Deployment Time	28-Aug-2004 15:04
Modification Time	28-Aug-2004 14:27
Version	1.2
Author	John Smith

DFDL grammar information
IBM App Connect Enterprise has only one compiled DFDL grammar for each Application domain.

A deployed Application has a level of isolation from anything else deployed. The Application sees only the DFDL schemas deployed within that Application.

- Anything outside of the Application cannot see the schemas inside the Application.
- Anything that is deployed just as a flow or message definition exists in its own space as well. Everything that is not explicitly in an Application can be thought of as in an Integration Server level Application domain.

Even though separate flows might use separate message models, each importing separate schema, when they are deployed, the schema/models are merged into a single model.

Unsupported features
The following features of the DFDL 1.0 specification are currently not supported in this implementation.

Property enumerations

The following property enumerations are unsupported.

Feature	**Reference in DFDL specification**
dfdl:textBiDi "yes"	21. Optional DFDL Features
dfdl:occursCountKind "stopValue"	16. Properties for Array Elements and Optional Elements

Feature	Reference in DFDL specification
dfdl:binaryCalendarRep "packed"	13.13 Properties Specific to Calendar with Binary Representation
dfdl:floating "yes"	14.4 Floating Elements
dfdl:lengthKind "endOfParent"	12.3 Properties for Specifying Lengths
dfdl:assert testKind "pattern"	7.3 The dfdl:assert Statement Annotation Element
dfdl:discriminator testKind "pattern"	7.4 The dfdl:discriminator Statement Annotation Element
dfdl:separatorSuppressionPolicy "trailingEmptyStrict"	14.2 Sequence Groups with Separators
dfdl:encodingErrorPolicy "replace"	11.2 Character Encoding and Decoding Errors
dfdl:textZonedSignStyle "asciiTandemModified"	13.6 Properties Specific to Number with Text representation
dfdl:bitOrder "leastSignificantBitFirst"	11. Properties Common to both Content and Framing
dfdl:escapeCharacterPolicy "delimiters"	13.2.1 The dfdl:escapeScheme Properties

Properties

The following properties are unsupported:

Feature	Reference in DFDL specification
dfdl:hiddenGroupRef	14.5 Hidden Groups
dfdl:inputValueCalc dfdl:outputValueCalc	17. Calculated Value Properties.

Functions

The following functions are unsupported in DFDL expressions:

Feature	Reference in DFDL specification
fn:count fn:exactly-one	23.5 Constructors, Functions and Operators
fn:name fn:local-name fn:namespace-uri	
dfdl:contentLength dfdl:valueLength	
dfdl:testBit dfdl:setBits	
dfdl:occursIndex	
dfdl:checkConstraints	

Feature	Reference in DFDL specification
dfdl:timeZoneFromDate dfdl:timeZoneFromDateTime dfdl:timeZoneFromTime	
All dfdl: constructor functions	
dfdl:decodeDFDLentities dfdl:encodeDFDLentities dfdl:containsDFDLentities	

Annotations

The following annotations and annotation placements are unsupported:

Feature	Reference in DFDL specification
dfdl:defineVariable with external=true	7.7 The dfdl:defineVariable Annotation Element
dfdl:newVariableInstance	7.8 The dfdl:newVariableInstance Statement Annotation Element
dfdl:assert on global element and simple type	7.3 The dfdl:assert Statement Annotation Element
dfdl:discriminator on global element and simple type	7.4 The dfdl:discriminator Statement Annotation Element

Other restrictions

The following restrictions also apply:

Feature	Reference in DFDL specification
Use of default values is not supported during parsing.	9.4 Element Defaults
When default values are applied to a missing complex element during serialization and the element is a choice, only the first choice branch is used to supply default values. If an error occurs on the first branch then serialization fails.	9.4 Element Defaults
When dfdl:lengthKind is 'implicit' for a complex element, the element can be prematurely terminated by encountering a higher level terminating delimiter.	12.3.2.1 Non-Delimited Elements within Delimited Constructs
When encoding is 'UTF-8', 'UTF-16' or 'UTF-32' a byte order mark at document start is not automatically processed, and must be modeled explicitly if it is to affect the encoding of the document.	11.1 Unicode Byte Order Marks (BOM)
Schemas must contain only one DFDL xs:appinfo element within each xs:annotation element, as subsequent xs:appinfo elements might not be processed correctly.	6. DFDL Syntax Basics

Feature	Reference in DFDL specification
When dfdl:lengthKind is 'prefixed', the simple type referenced by the dfdl:prefixLengthType property can not also have dfdl:lengthkind 'prefixed'.	12.3.4 dfdl:lengthKind 'prefixed'
Not all instances of inapplicable DFDL property placement are reported.	2.7 Optional Checks and Warnings
All global elements are assumed to be potential document roots and are validated accordingly.	2.7 Optional Checks and Warnings
dfdl:lengthKind 'pattern' can not be used with elements of binary representation.	12.3.5 dfdl:lengthKind 'pattern'
When parsing, the distinction between an element being 'missing', having an 'empty representation' and having an 'absent representation', is not in accordance with the specification.	9.2 DFDL Data Syntax Grammar
When an element has binary representation, a DFDL character entity is not allowed in the definition of a nil value for the element.	13.15 Nil Value Processing
dfdl:textStandardDecimalSeparator is not allowed to be a list of values.	13.6 Properties Specific to Number with Text representation
dfdl:textStandardExponentRep is not allowed to be more than one character, and not allowed to be empty string.	13.6 Properties Specific to Number with Text representation
The '@' (significant digits) symbol is not allowed in dfdl:textNumberPattern.	13.6 Properties Specific to Number with Text representation
The '_' (underscore) character is not allowed in dfdl:calendarLanguage.	13.11 Properties specific to Calendar with Text or Binary Representation
dfdl:calendarTimeZone is not allowed to be empty string or to be an Olson format time zone.	13.11 Properties specific to Calendar with Text or Binary Representation
A binary packed number with dfdl:lengthUnits 'bits' is not checked to see that the length is a multiple of 4 bits.	12.3.7.2.6 Length of Packed Decimal Calendar Elements
A binary packed number with dfdl:alignmentUnits 'bits' is not checked to see that the alignment is a multiple of 4 bits.	12.1.3 Mandatory Alignment for Packed Decimal Data
All elements with text representation are assumed to align on an 8-bit boundary. There is no support for encodings that are not 8-bit aligned.	12.1.2 Mandatory Alignment for Textual Data

Feature	Reference in DFDL specification
dfdl:textStandardDecimalSeparator, dfdl:textStandardGroupingSeparator, dfdl:textStandardExponentRep, dfdl:textStandardInfinityRep, dfdl:textStandardNanRep and dfdl:textStandardZeroRep are not checked to ensure they are all distinct from one another.	13.6 Properties Specific to Number with Text representation
dfdl:calendarLanguage is not allowed to be an expression.	13.11 Properties specific to Calendar with Text or Binary Representation
dfdl:assert message is not allowed to be an expression.	7.3 The dfdl:assert Statement Annotation Element
dfdl:discriminator message is not allowed to be an expression.	7.4 The dfdl:discriminator Statement Annotation Element
Path locations in DFDL expressions are not correctly validated. Specifically, array elements without predicates and references into other choice branches are not flagged as errors. Avoid using such path locations.	23. Expression language
dfdl:initiator and dfdl:textStandardZeroRep incorrectly allow the use of %WSP*; entity class on its own. Use %WSP+; instead.	12.2 Properties for Specifying Delimiters , 13.6 Properties Specific to Number with Text representation
dfdl:terminator and dfdl:separator incorrectly allow the use of %WSP*; entity class on its own when scanning for delimiters. Use %WSP+; instead when scanning for delimiters.	13.6 Properties Specific to Number with Text representation , 14.2 Sequence Groups with Separators
When serializing, the application of an escape scheme takes place after the application of dfdl:emptyValueDelimiterPolicy.	13.2.1 The dfdl:escapeScheme Properties
A value of empty string for dfdl:binaryBooleanTrueRep is not supported.	13.10 Properties Specific to Boolean with Binary Representation
The fn:concat function is limited to four arguments.	23.5.2.3 String Functions

Implementation-specific limits
Some limits in the DFDL specification are implementation-defined.

The following limits apply to this implementation:

- A 1,000 digit limit applies during parsing or serializing of numbers with text representation.
- A maximum precision of 33 digits is supported during parsing or serializing of numbers. If this limit is exceeded, behavior is undefined.
- The maximum number of elements in an array is 2,147,483,647.
- Fractional second digits are supported to millisecond accuracy.

Message Sets: Message sets overview

A message set is the original container for message models used by IBM App Connect Enterprise.

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

A message set is a folder in a message set project that contains a logical grouping of your messages and the objects that comprise them (elements, types, groups). A message set contains the following files:

- Exactly one message set file
- Zero or more message definition files
- Zero or more WSDL files
- Zero or more message category files

Note: A message set cannot contain JSON Schema.

The message set file provides message model information that is common to all the messages that are defined in the message set. You can create this information by using the Message Set editor.

You can base a new message set on an existing message set. In this case, all the definitions in the existing message set are copied into the new message set.

When you have created your message set, you must specify the following key properties:

Supported message domains
: The message domains that are supported by the message set. The supported domains determine what is generated for deployment to an integration node, and are used when parsing and writing the messages that are defined within the message set.

Default message domain
: The default domain of the message set.

When you have created your message set, you typically import application message formats described by XML DTD, XML Schema, WSDL files, C structures, COBOL structures, EIS systems, or creating and populating message definition files. Using the Message Definition editor, you can then edit the logical structure of your messages, and create and edit the physical formats that describe the precise appearance of your message bit stream during transmission. Alternatively, you can create an empty message definition file and create your messages by using only the editor.

When your message definition files are complete, you can generate the message set in a form that can be used by an integration node, parser, or application. This might be in one of the following forms:

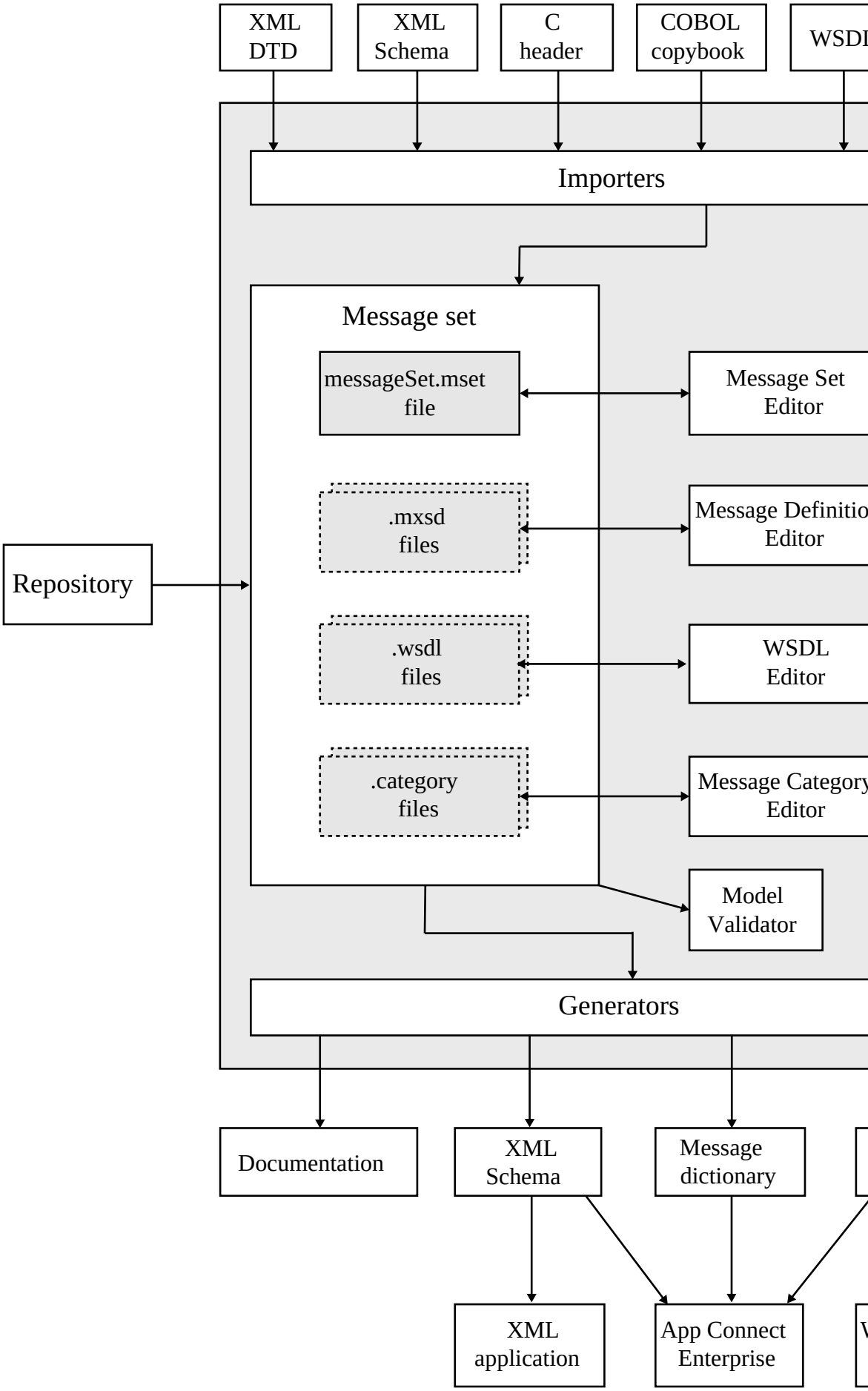
- A message dictionary for deployment to an integration node
- An XML Schema file for use by an application to validate XML messages, or for deployment to an integration node
- A Web Services Description Language (WSDL) file for a web services client, or for deployment to an integration node
- Documentation to give to programmers or business analysts

Optionally, messages can be grouped into message categories for convenience. You can add messages to message categories by using the Message Category editor.

Each time you save a message set file, message definition file, or message category file, the content is validated to ensure that the message model that you are creating follows certain rules. There are rules for both the logical structure and the physical formats. This model validation ensures the integrity of your model, but does not necessarily prevent you from saving a message model file that is not valid.

IBM App Connect Enterprise supplies a range of parsers to parse and write message formats. Each parser is suited to a particular class of messages (for example, fixed-length binary, delimited text, or XML) known

as a message domain. When you create a message set, you specify which domains the message set supports. This property determines which parsers can be used when you parse and write messages that are defined within that message set.


XML
DTD
XML
Schema
C
header
COBOL
copybook
WSDL
Importers
Message set
messageSet.mset
file
Message Set
Editor
.mxsd
files
Message Definitio
Editor
Repository
.wsdl
files
WSDL
Editor
.category
files
Message Category
Editor
Model
Validator
Generators
Documentation
XML
Schema
Message
dictionary
XML
application
App Connect
Enterprise

Message Sets: Message set projects
A *message set project* is a specialized container in which you create and maintain all the resources associated with one message set.

Tip: Message set projects are used when working with message sets. A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

The content of a message set project is a single message set folder, and additional folders if you are modeling EIS data, CORBA data, or SCA import or export data. The name of the message set project provides the name of the message set. You can create a message set project by using the New Message Set wizard.

The following restrictions apply to message set projects:

- A message set project must contain just one message set.
- A message set project cannot refer to any other message set.

Message Sets: Message set resources
Resources in a message set are created as files, and are displayed under the message set folder in the Application Development view.

- Message set file `messageSet.mset`

  There is always one, and only one, `messageSet.mset` file in a message set. This file contains message model properties that are common to all the content of the message set. It is also where you define the physical formats that you want for this message set. These can be Custom Wire Format (CWF), Tagged Delimited String Format (TDS), and XML Wire Format (XML).

  The file is created for you when a new message set is created, and you manipulate its content with the Message Set Editor.
- Message definition files that have the suffix `.mxsd`

  You can have many message definition files in a message set. Each file contains the logical model and the associated physical model, in XML Schema form, for a group of related messages.
- Deployable WSDL files that have the suffix `.wsdl`

  These files are used by the SOAP domain. You can have many WSDL files in a message set.
- Message category files that have the suffix `.category`

  These files are optional. You can have many message category files in a message set. A message category provides another way of grouping your messages, perhaps for documentation purposes.

When you have completed the resources in your message set, you can generate the content of the message set in a form that can be used by an integration node parser or an application. This might be:

- a message dictionary for deployment to an integration node
- XML Schema for use by an application building XML messages, or for deployment to an integration node
- Web Services Description Language (WSDL) for a web services client, or for deployment to an integration node
- documentation to give to programmers or business analysts

Message Sets: Message set identification
A message set is identified by the name of the message set folder in the message set.

When you must refer to a message set from a message flow (for example, when setting the `Message model` property of an input node), use the message set name.

A message set also has a 13-character identifier that is guaranteed to be unique. You can use this identifier, instead of the message set name, to refer to a message set, but only if you are using the MRM or IDOC domains. Other domains do not recognize the identifier.

A message set also has an alias. An alias can be used only with MRM multipart messages.

Message Sets: Message set limitations
A message set is the original container for message models used by IBM App Connect Enterprise.

Tip: In WebSphere Message Broker Version 8.0, message model schemas in integration projects are the preferred way to model messages for all kinds of data format. Message sets continue to be supported, and are required if you use the MRM or IDOC domains for your formats.

You can have as many message definition files as you want within one message set, but you must limit your message sets to a few related message definition files that share the same physical formats.

There are several reasons for the suggested limitations:

- Generation of a message dictionary and other representations is quicker.
- Generated documentation is more manageable.
- MRM physical formats apply to *all* objects within the message set.

  Therefore, for example, if you are using the MRM domain and have an XML message and an unrelated CWF message modeled in the same message set, CWF physical format properties are present for all objects. But the CWF properties are of no interest to the XML message and therefore take default values in those objects, which can result in unwanted task list warnings.
- You cannot use recursion for MRM CWF and TDS physical formats.

  Therefore, if you are modeling XML messages that have a recursive structure, you must ensure that recursive XML messages do not share a message set with MRM CWF or TDS physical formats.

Message Sets: Message set version and keywords
When you develop a message set, you can define the version of the message set, and other key information that you want to be associated with it.

After you have deployed the message set in a BAR file, you can view the message set properties in the IBM App Connect Enterprise Toolkit and the web user interface. The properties include the deployment and modification dates and times (the default information that is displayed), and the additional version or keyword information that you have set.

Version

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

You can also define a default message set version in the `Default version tag` of the message set preferences. All message sets that you create after you have set this property have this default applied to the `Version` property at the message set level.

Keywords

You must define keywords in the `Documentation` property of the message set. Keywords follow certain rules to ensure that the information can be parsed. The following example shows the type of information that you can define in the `Documentation` property:

```
$MQSI Author=John Smith MQSI$
```

The following table contains the information that is displayed by the IBM App Connect Enterprise Toolkit and the web user interface:

Message set name	
Deployment Time	28-Aug-2004 15:04

Modification Time	28-Aug-2004 14:27
Version	1.0
Author	John Smith

Message Sets: Message definition files
A *message definition file* contains the messages, elements, types, and groups which make up a message model within a message set.

Tip: Message definition files are used when working with message sets. A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

Every message set requires at least one message definition file to describe its messages. Message definition files use the XML Schema language to describe the *logical format* of one or more messages. Extra information in the form of XML Schema annotations is used to describe any *physical formats* that you define for the messages.

Large message sets can contain several message definition files. This keeps the individual files to a manageable size, making them faster and easier to work with.

Message definition files can be created by using the Message Definition editor, or can be imported from a range of different file formats as described in "Importing from other model representations to create message definitions" on page 2139.

A message definition file can be associated with a namespace, so that all message model objects that are declared within the file belong to that namespace. Namespaces provide a means of avoiding name clashes among similarly named global objects. They are described in detail in "Namespaces in the message model" on page 2159.

One message definition file can reuse message model objects that are defined in another message definition file. XML Schema provides two mechanisms to do this: **`import`** and **`include`**. For more information, see "Reusing message model files " on page 2161.

Message Sets: XML Schema restrictions in message sets
Some XML Schema 1.0 features are not supported when message models are contained in message sets.

Unsupported XML Schema features

The following feature is accepted, but not supported, and causes validation errors if it is used in your message model:

- Redefines

A quick fix is provided to convert Redefines into Include.

Further information about XML Schema

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Message Sets: XML Schema extensions in message sets
IBM App Connect Enterprise provides some additional facilities that are not specified in the XML Schema 1.0 specification. When using a message set, further extensions are provided.

Composition

The message model in a message set adds the following compositions that are beyond the XML Schema 1.0 specification:

message
A refinement of *choice* that can contain only a set of references to messages within the same message set. Groups and complex types with composition of *message* are used when modeling multipart messages.

orderedSet
A set of elements that must be present in the order that they are listed. Groups cannot be used within an *orderedSet*. Elements can repeat, but duplicate elements cannot be used.

unorderedSet
A set of elements that can be present in any order. Groups cannot be used within an *unorderedSet*. Unlike an *all* group, elements within an *unorderedSet* can repeat. However, duplicate elements cannot be used.

Compositions *orderedSet* and *unorderedSet* enable message models that were produced in earlier versions of the product to be supported.

Content validation

The message model in a message set adds a validation control called ‘Content validation' that is used only if the domain is MRM or IDOC, and if validation is selected. It determines how strictly the content of the group is validated. See MRM content validation for more details.

The Content validation property does not affect validation in the XMLNSC or SOAP domains. Validation in these domains follows the rules of XML Schema 1.0.

Physical format information

If one or more physical formats are defined for a message set, the XML Schema objects within the message set can hold extra information about how they must be parsed and serialized.

Further information about XML Schema

For details about XML Schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Message Sets: Multipart messages
A *multipart message* contains one or more other messages within its structure. The contained message is sometimes referred to as an *embedded message*.

Tip: Multipart messages are used when working with messages that are modeled by using message sets. A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see “Enabling message set development” on page 2248 .

A multipart message must contain a group, or a complex type, with its `Composition` property set to Message. This group or complex type can contain a list of references to messages that can be present at that location in the message structure. If the group or complex type is empty, any message can be present. When a message is parsed, only one embedded message can be present in that location.

Message envelopes

A common use of multipart messages is to define an outer message with a fixed structure. This outer message is called the *message envelope*. Within the message envelope a group or complex type is included, as described earlier in this topic. Examples of message standards that can be modeled by using this technique are EDIFACT, X12, SWIFT, SOAP XML, SAP ALE IDoc, multipart MIME, and RosettaNet.

Identifying the embedded message

When a multipart message is parsed, the parser must be able to identify the embedded message; it might be any of the messages that are referenced by the group or complex type, or it might be a message that

is not referenced by the group or complex type, perhaps from a different message set. This is achieved by using one of four techniques, *Automatic*, *Message Identity*, *Message Path*, or *Manual*.

Automatic
Used when parsing XML messages, such as SOAP. The parser automatically identifies and parses embedded messages by using the tag in the XML document.

Message Identity
Used by the MRM parser. See "Message Sets: Identifying an embedded message by using a Message Identity" on page 2176.

Message Path
Used by the MRM parser. See "Message Sets: Identifying an embedded message by using a Message Path" on page 2178.

Manual
Used by the MIME parser. The parser treats embedded messages as BLOBs. If you want to parse the BLOB by using another parser, you must do so manually by using ESQL, or Java, or a ResetContentDescriptor node.

Restrictions

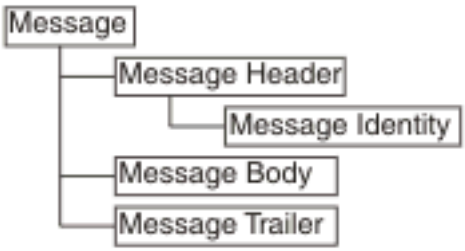
Unless using the *Manual* identification technique, all embedded messages must be of the same physical format as the outermost message, and have the same character set and encoding.

When using the *Automatic* or *Message Path* identification techniques, all embedded messages must be from the same message set as the multipart message.

Message Sets: Identifying an embedded message by using a Message Identity
You can identify an embedded message by using the Message Identity. This technique is used by the MRM domain, and replaces the use of the Message Key.

Tip: Multipart messages are used when working with messages that are modeled by using message sets.

The Message Identity technique for identifying embedded messages is useful when a multipart message has a format such as that shown in the diagram:



In this example, the Message Header and Message Trailer act as an envelope for the message body. They typically have a fixed structure, although the Message Body can be defined with many different structures.

A place holder for an embedded message is created by setting the `Composition` property of the complex type or group of the Message Body element to Message. This enables an embedded message to be added within the outer Message, creating a multipart message.

When using the Message Identity technique to parse such a multipart message, the embedded message must be identified earlier in the Message Header by using a Message Identity element. A Message Identity Element is a string element (or attribute) that precedes the embedded message in the model and whose `Interpret Value As` property is set to Message Identity.

When a multipart message is input to a message flow, the Message Identity element must have a value that corresponds to either the `Name` or the `Message Alias` of the next embedded message in the bit stream. This enables the MRM parser to correctly identify the embedded message in the model.

For cases where the Message Identity element value does not match the `Name` of the message, use the `Message Alias` property to specify this value. The MRM parser tries to match on `Name` first, and if that fails, it tries to match on `Message Alias`.

When the MRM parser has encountered a Message Identity element, its value applies to all embedded messages that are contained immediately within the current message. This does not apply to embedded messages within embedded messages; any embedded message must have its identity provided by a Message Identity element within its immediate parent message.

If a second Message Identity element is encountered within the current message, its value overrides any previously held. This enables different peer embedded messages to exist within a given message.

Message Identity takes priority over Message Path. If both are specified, Message Identity is used. Use only one of these techniques for a given multipart message.

Embedded messages defined in different message sets

By default, an embedded message is assumed to be defined within the same message set as the current message. This can be overridden by using a Message Set Identity, which works in a similar manner to a Message Identity.

An embedded message that is defined within a different message set must have its message set identified earlier in the message, by using a Message Set Identity element. A Message Set Identity Element is a string element (or attribute) that precedes the embedded message in the model and whose `Interpret Value As` property is set to Message Set Identity.

When a multipart message is input to a message flow, the Message Set Identity element must have a value that corresponds to either the `Identifier`, `Name`, or `Message Set Alias` of the message set that defines the next embedded message in the bit stream. This enables the MRM parser to correctly identify the message set to use.

If the Message Set Identity element value does not match the `Identifier` or `Name` of the message set, use the `Message Set Alias` property to specify this value. The MRM parser tries to match on `Identifier` first, then on `Name`, and finally on `Message Set Alias`.

After the MRM parser has encountered a Message Set Identity element, its value applies to all embedded messages that are contained within the current message. It also applies to embedded messages within embedded messages, unless an embedded message also contains a Message Set Identity element.

If a second Message Set Identity element is encountered within the current message, its value overrides any previously held. This enables peer embedded messages to be contained within different message sets.

The following example of an X12 message shows the use of both Message Identity and Message Set Identity. The field that contains `004010X092` within the GS segment on line 0002 holds the Message Set Identity as a `Message Set Alias`. The 207 on line 0003 in the ST segment is the Message Identity held as a `Message Alias`. The embedded message is from line 0004 - 0015 inclusive.

Note: The line numbers and spaces at the beginning of each line are for illustrative purposes only and do not exist in the actual message.

```
0001  ISA*00*          *00*          *30*12-3456789     *ZZ
      *9876543-21     *000104*1820*U*00401*000000001*0*T*:!
0002  GS*HS*HOSP CLAIM*PAYER ADJDEPT*20000104*1820*1*X*004010X092!
0003  ST*270*1234!
0004  BHT*0022*13*10001234*19990501*1319!
0005  HL*1**20*1!
0006  NM1*PR*2*ABCCOMPANY*****PI*842610001!
0007  HL*2*1*21*1!
0008  NM1*1P*2*BONE AND JOINT CINIC*****SV*2000035!REF*N7*234899!
0009  N3*55*HIGH STREET!
0010  N4*SEATTLE*WA*98123!
0011  HL*3*2*22*0!TRN*1*93175-12547*9877281234!
0012  NM1*IL*1*SMITH*ROBERT*B***MI*11122333301!
0013  REF*1L*599119!
0014  DMG*D8*19430519*M!
0015  DTP*472*RD8*19990501-19990515!EQ*30**FAM!SE*17*1234!
0016  GE*1*1!IEA*1*000000001!
```

Physical format considerations

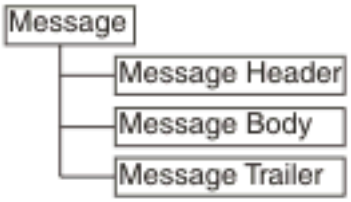
Both Message Identity and Message Set Identity are applicable to all physical formats.

Message Sets: Identifying an embedded message by using a Message Path
The Message Path technique for identifying embedded messages is useful when the multipart message contains no information about the identity of an embedded message.

Tip: Multipart messages are used when working with messages that are modeled by using message sets.

This technique is used by the MRM domain.

In the diagram, the Message Header and Message Trailer act as an envelope for the message body. Typically, they have a fixed structure, but the Message Body can be defined with many different structures.



A place holder for an embedded message is created by setting the `Composition` property of the complex type or group of the Message Body element to Message. This enables an embedded message to be added within the outer message, creating a multipart message.

When using the Message Path technique to parse such a multipart message, the embedded message must be identified by a fixed path to the innermost message from the outermost message. For this example, this would be:

```
Message/Message Body
```

If the path to the innermost message contains intermediate elements, these intermediate elements must also be included in the path. In the following example, these elements are shown in bold:

```
Message/Data1/Data12/Message Body
```

This technique can be used to identify nested embedded messages as well, by extending the path. For example:

```
Message/Data1/Data12/Message Body/Data2/Inner Message
```

The path is specified by using one or both of two properties, the `Message Type` property of an IBM App Connect Enterprise input node (or MQRFH2 header) and the `Message Type Prefix` property of the containing message set. These two properties are combined to produce a final path that is used to locate embedded messages.

Message Identity takes priority over Message Path. If both are specified, Message Identity is used. Use only one of these techniques for a given multipart message.

You cannot use the Message Path technique to identify multiple peer embedded messages.

Embedded messages defined in different message sets

This option is not supported by the Message Path technique.

Physical format considerations

The Message Path technique is applicable to all physical formats.

Message Sets: Message categories
Message category files have the suffix `.category`. These files are optional. You can have many message category files in a message set.

A message category provides another way of grouping your messages, perhaps for documentation or convenience purposes. A message category can also be used for assisting in the generation of Web Services Description Language (WSDL) files, but this use is deprecated. For more information, see "Generation of WSDL files" on page 2206.

A message set category file is created by using the New Message Category File wizard.

When you have created your message category file, you must specify one key property.

Message Category Kind
The Message Category Kind indicates whether the message category is to participate in the generation of WSDL files.

You can then add messages to the message category file. If the message category is to participate in WSDL generation, assign appropriate values to the `Role Name` and `Role Type` properties of each member message.

Message category identification

The name of a message category is provided by the name of the `.category` file. You can change the message category name by renaming the file.

Message Sets: Physical formats in the MRM domain
Each message definition file within a message set describes both the *logical structure* of your messages, and the *physical formats* that describe the precise appearance of your message bit stream during transmission.

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

If you are using the MRM domain or the IDOC domain, physical format information *must* be provided, as it tells the parser exactly how to parse the message bit stream.

You can think of a message as a packet of data that is sent from one place to another. The sender and receiver of the message will have agreed the structure of the message and what each field in the message means. This is the platform and protocol independent logical structure.

The sender and receiver will have also agreed on the physical representation of the message, how the data is physically laid out on the wire. For example, if you define a message that conveys information about a debit of an individual's bank account, it can be represented in different physical forms (examples are XML, or a fixed structure such as a COBOL copybook). The meaning and data are the same in both cases: only the physical layout has changed.

If you are using the MRM domain, you can model various different physical representations by using named physical formats.

- Use the Custom Wire Format (CWF) physical format to model fixed-format messages from applications that are written in C, COBOL, PL/I and other languages. This support includes the ability to create a message model directly from a C header file or COBOL copybook.
- Use the Tagged Delimited String Format (TDS) physical format to model formatted text messages, perhaps with field content identified by tags or separated by specific delimiters or both. This support is rich enough to model industry standards such as SWIFT, EDIFACT, and X12.
- Use the XML physical format to model XML messages, including those that use XML namespaces. This support includes the ability to create a message model directly from an XML DTD or XML Schema file.

Different physical representations

The following example shows how a simple logical message can have different physical representations.

The logical model defines the structure and order of the message. In the following example, the three fields are simple integers, and C follows B, which follows A:

```
int   A;
int   B;
int   C;
```

- A typical Custom Wire Format representation for this logical message would be 12 bytes of data, with each of A, B and C occupying 4 bytes. Alternatively, perhaps A is 4 bytes long, but B and C are only 2 bytes long. You supply the precise physical information for each field in the message as CWF properties.
- TDS enables several different representations to be modeled. Each integer can be preceded by a tag to identify it and a delimiter to terminate it, as follows:

  ```
  {A_tag:xxxxxxxx;B_tag:xxxxxxxx;C_tag:xxxxxxxx}
  ```

  An alternative might rely on the data being ordered so only the terminating delimiter must be specified, as follows:

  ```
  [xxxxxxxx;xxxxxxxx;xxxxxxxx]
  ```

  You supply the precise identification regime as TDS properties.
- A typical XML representation of this model is as follows:

  ```
  <Msg><A>xxxxxxxx</A><B>xxxxxxxx</B><C>xxxxxxxx</C></Msg>
  ```

  where xxxxxxxx is the value of the integer represented as a string (XML deals only with strings). An alternate representation might be:

  ```
  <Msg A="xxxxxxxx" B="xxxxxxxx" C="xxxxxxxx"/>
  ```

  where the values of the integers are stored as XML attributes rather than XML elements. You supply the precise XML rendering for each field in the message as XML properties.

This shows that the logical model is unchanged. It is constant, regardless of the physical representation that you choose to model on top of it, using the physical format support provided by the MRM domain. The MRM parser is able to transform the message from the input physical representation to any number of output representations, based on the physical format layers that you have defined.

Creating physical formats

When you have created your message set, you can create physical formats. You do this using the Message Set Editor. When you next save the `messageSet.mset` file, any new physical formats are added to all the objects in all the message definition files in that message set.

The next time you edit an object in a message definition file, you see the physical formats in the properties hierarchy pane of the Properties tab. If you click a physical format for an object, you are presented with a property sheet where you can enter the information for that physical format for that object.

Not all objects have properties in all physical formats. For example:

- CWF properties apply only to local elements and attributes, and element and attribute references.
- Complex types and groups have only TDS properties.
- Messages have only XML properties.

These differences occur because of the different nature of each physical format, and are explained in more detail in the section for each physical format.

There is no limit to the number of physical formats you can create in a given message set. However there are some recommendations that apply if you want to mix physical formats of different kinds in the same message set.

Physical formats can be deleted if no longer required.

MRM Custom Wire Format
Custom Wire Format (CWF) is the physical representation of a message that is composed of a number of fixed-format data structures or elements, which are not separated by delimiters.

Tip: In Version 8, when modeling and parsing general text or binary data use message model schema with the DFDL domain, instead of message sets and the MRM domain.

Within a CWF messaging environment, it is not possible to distinguish one element from the next without knowledge of the message structure. To correctly determine the values of individual elements, the following information must be made available to the message parser:

- The order (this is defined in the logical properties)
- The length (can be specified in bytes, characters, or character units)
- The cardinality (that is, the number of repeats)
- The type of data contained in each element (this is partly defined in the logical properties but can be further qualified in the CWF physical format)
- A number of characteristics based upon the logical type of the data

A CWF physical format is typically used to describe messages which are mapped to a C structure, a COBOL copybook, or other programming language data structure definition.

You can add more than one CWF physical format to a message set, but within that message set, each physical format must have a unique name. When parsing a CWF message by using the MRM parser, the physical format name specifies the physical properties that are to be used by the parser.

Adding a CWF physical format to a message set enables you to process input messages and construct output messages in this format. Messages can be transformed between CWF and the other physical representations (for example TDS or XML). While the other physical representations support self-defining elements (that is elements which do not have a definition in the logical model) within the MRM domain, the parsing of a CWF message does not. Consequently, any such self-defining elements are discarded during the output of messages in CWF format.

MRM Custom wire format: Message model integrity
When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called *model validation*.

The CWF physical format depends on fixed-format data structures. Therefore, most tests that are applied to a CWF message confirm that each fragment of a message - and therefore, the message as a whole - has a well-defined length. Therefore, these tests examine properties such as `Length`, `Length Reference` and `Length Units`.

Typically, one or other of `Length` and `Length Reference` must be set. If `Length Reference` is set, it must refer to an element that is of simple type integer and that appears earlier in the message than the current item.

Tests other than these tend to be both simple and obvious so that, for example, the message set property `First Day of Week` must be the name of a day in the week.

The fact that CWF relies on fixed-format data structures also imposes some limitations on the messages that can be represented:

- CWF cannot represent a message that includes the use of XML Schema wild cards; this is a consequence of its inability to handle undefined content.
- CWF cannot represent a message that includes recursive definitions.
- CWF cannot represent a message that includes the use of substitution groups, because there is no way to recognize the substituted element.

MRM Custom wire format: NULL handling
CWF supports the handling of explicit NULL values within messages, if the logical nillable property of the element is set.

An explicit null is identified by a specific value that identifies an element as being null.

The `Boolean Null Value` can be specified at the message set level, and applies to the Boolean elements of all messages that are defined in that message set. The null value of all other element types can be specified individually for each element.

CWF supports the representation of null values by using the `Encoding Null` and `Encoding Null Value` element properties. Together, this information controls how null values are handled by the MRM parser.

The `Encoding Null` property can be set to one of four values:

NullLogicalValue

The `Encoding Null Value` property is interpreted as a logical value. Therefore, if its value is set to 0, for example, both `0` and `0.00` are interpreted as null values.

NullLiteralValue

The `Encoding Null Value` property is interpreted as a string value. Therefore, the value of the element in the bit stream must match exactly the value that is specified to be interpreted as a null value.

NullPadFill

Used for fixed-length elements. On output, any element with a null value is padded to the appropriate length with the specified padding character.

NullLiteralFill

The `Encoding Null Value` property is interpreted as a single character string value. Therefore, each character of the value of the element in the bit stream must match exactly the character value specified to be interpreted as a null value.

MRM Custom wire format: Multipart messages
The Custom Wire Format (CWF) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

Alternatively, you can resolve an embedded message by using ESQL or Java to identify the message. The first message that you reference in this way is assumed to be the selected message. This technique works in an identical manner to `unresolved choice` handling.

MRM Custom wire format: Data Conversion
The Custom Wire Format supports the conversion of data to a different code page (for string simple types) or encoding (for numeric simple types), or both.

A message set contains properties to enable the character (CCSID) and numeric encoding (`Byte Order` / `Float Format`) information to be specified. If you generate a message dictionary for deployment to an IBM App Connect Enterprise, this information can be overridden by using the appropriate fields of the IBM MQ message header, or other transport header.

MRM Custom wire format: Relationship to the logical model
Some restrictions exist in relation to the logical model for messages that are defined by using the CWF.

Composition

A CWF message is always written with the elements in the sequence that is specified in the logical message model definition. However, you do not always have to specify the ESQL or Java that builds the elements in that sequence. The following rules for coding ESQL are given for each value of the type `Composition` property.

Sequence
You must build the output message to match the sequence of the elements or groups in the message. You can normally do this using ESQL SET statements to assign a value to each element or type. The first SET statement sets the value of the first element or type in the message, the second SET statement sets the value for the second element or type, and so on. You can vary this sequence of statements by using ESQL ATTACH, CREATE, and MOVE statements.

If the elements or types have default values, and you do not build the message in the correct sequence, those elements that are built out of sequence contain their default values, not the values that you set. This is because elements that are built out of sequence are assumed to be self-defining and, for CWF, these are discarded when the message is written to the bit stream.

Ordered Set
You must build the output message to match the sequence of the elements in the message. You can normally do this using ESQL SET statements to assign a value to each element. The first SET statement sets the value of the first element in the message, the next SET statement sets the value for the second element, and so on. You can vary this sequence of statements by using ESQL ATTACH, CREATE, and MOVE statements.

If the elements have default values, and you do not build the message in the correct sequence, those elements that are built out of sequence contain their default values, not the values that you set. This is because elements that are built out of sequence are assumed to be self-defining and, for CWF, these are discarded when the message is written to the bit stream.

Unordered Set
You can build elements of the output message in any sequence. On output, the elements are written in the order that is specified in the logical message model definition.

All
You can build elements of the output message in any sequence. Each element must be present only once (that is, it must not repeat). On output, the elements are written in the order that is specified in the logical message model definition.

Choice
A choice cannot be resolved purely from the data. The receiving program must interpret the data and decide which option of the choice the message instance contains. This process is known as *unresolved choice* handling. The first reference in the application to any one of the choice elements resolves the choice to the option that contains that element.

Message
Mechanisms for the resolution of embedded messages are discussed in the "MRM Custom wire format: Multipart messages" on page 2182 topic.

Content validation

CWF is a fixed format, and all elements must be present in a message. Therefore, content validation is ignored. On output, all elements must be set explicitly (for example, by using ESQL SET), set implicitly (by using a tree copy function), or must have a default value defined.

Default values

On output of a CWF message in the MRM domain, any element, or occurrence of an element for which a value has not been set (either explicitly or implicitly), inherits the specified default value of the element. If no default value has been specified then an exception is thrown.

Min Occurs and Max Occurs

The logical properties `Min Occurs` and `Max Occurs` specify the permitted number of occurrences of an element, or group, in a message. These properties are used when parsing and writing messages, and when validating the content of a message.

In CWF, `Max Occurs` occurrences are expected when parsing, and `Max Occurs` occurrences are produced when writing. Default values are used for missing elements, and any excess elements are discarded.

- A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is ignored, `Max Occurs` is assumed.
- Optional occurrence (`Min Occurs` = 0) is ignored, `Max Occurs` is assumed.
- Always absent (`Max Occurs` = 0) is allowed.
- An unbounded number of occurrences (`Max Occurs` = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by the end of the message bit stream. On writing, the writer writes all occurrences in the message tree, if this number is less than `Min Occurs`, additional default values are written.

These rules arise because, in a CWF message format, there are no tags or other markup that can be used to determine the end of a variable number of repeats.

However this behavior is overridden if the CWF property `Repeat Reference` is set, which indicates that the number of occurrences is given instead by an integer element that occurs earlier in the message. In this case `Max Occurs` is ignored.

When validating, `Min Occurs` and `Max Occurs` are both used to check that the content of the message tree matches the model.

Simple types - lists and unions

Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union causes a task list warning if a CWF physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. The dictionary generator omits messages defined to contain such elements or attributes from the CWF section of the dictionary.

MRM TDS format

The Tagged/Delimited String format (TDS) is the physical representation of a message that has a number of data elements separated by tags and delimiters.

Tip: In Version 8, when modeling and parsing general text or binary data, use message model schema with the DFDL domain, instead of message sets and the MRM domain.

The TDS physical format is designed to model messages that consist of text strings, but it can also handle binary data. Examples of TDS messages are those that conform to the ACORD AL3, EDIFACT, HL7, SWIFT, or X12 standards. The TDS physical format allows a high degree of flexibility when defining message formats, and is not restricted to modeling specific industry standards; therefore, you can use the TDS format to model your own messages.

TDS message characteristics

There are a number of features of text string messages that are common to many formats. This is an overview of the main features that are supported by the TDS physical format:

Tags

The text strings in the message can have a *tag* or a label preceding the data value. The tag is a string that uniquely identifies the data value. The TDS format allows you to associate a tag with each element when you define the element.

Delimiters and tagged data separators

The message can contain various special characters or strings in addition to the tags and text string data values. The TDS format supports a number of different types of special characters or strings.

Some messages have a special character or string that separates each data value from the next. In the TDS format this is a known as a *delimiter*.

In formats that have a tag before each data value, the tag can be separated from its data value by a special character or string. In the TDS format this is known as a *tag data separator*.

Group indicators and terminators

A message can be split into a number of substructures in a similar manner to a COBOL or C structure. You can model each of these substructures separately by defining groups, complex types, or elements for each one.

A substructure can have a special character or string that indicates its start within the data. This is known in the TDS format as a *group indicator*.

A substructure can also have a special character or string that indicates its end in the data. In the TDS format, this is known as a *group terminator*.

A group indicator and group terminator can also be defined for the whole message. Group indicators and group terminators are optional for the message and each substructure.

Fixed-length strings
Some text strings within a message can be of fixed length; therefore, a delimiter between each data value is not necessary. This is supported by the TDS format.

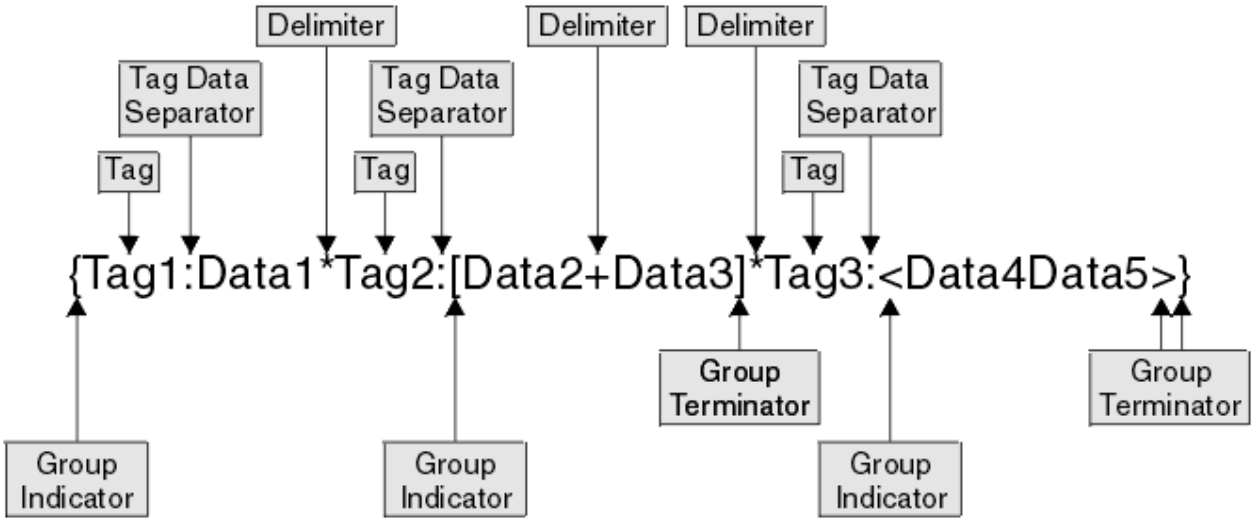
Fixed-length tags
Some tags can be defined as fixed length; therefore, a tag data separator is not necessary.

Separation types
The TDS property that controls the way text strings are separated is `Data Element Separation`. It has several options, for example, whether tags are used, whether strings lengths are fixed or variable, and what types of text strings are permitted.

The substructures within a message can use different types of data element separation and use different special characters. Therefore the TDS format allows you to define different types of data element separation and special characters for each complex type within the message.

Regular Expressions
If you choose the Use data pattern option for `Data Element Separation`, you can use regular expressions to identify parts of the message data to be assigned to sub fields. This is done by setting the regular expression in the `Data Pattern` property.

The following diagram shows an example data message with each of its components labeled.



- At the top level, each data value has a *tag* associated with it, each tag is separated from its data value by using a *tag data separator* of colon (:), and the data values are separated from each other using the asterisk *delimiter* (*).
- The *group indicator* for the message is the left brace ({) and the *group terminator* is the right brace (}).
- The data values `Data2` and `Data3` are in a substructure in which there are no tags, and each data element is separated from the next using the plus delimiter (+). The group indicator for this substructure is the left bracket ([) and the group terminator is the right bracket (]).
- The data values `Data4` and `Data5` are in a substructure in which the values are fixed length, and are therefore not separated by a delimiter. The group indicator for this substructure is the less than symbol (<) and the group terminator is the greater than symbol (>).

MRM TDS format: Determining the length of simple data values
The TDS format supports two categories of simple data types: textual and non-textual.

The Physical Type of an element determines whether it is categorized as textual or non-textual.

Textual data
Physical Type is either Text or TLOG Specific. For textual data, the Data Element Separation of the parent complex type or group determines how the length of the data is determined. See "MRM TDS format: Data element separation" on page 2186 and its subtopics.

Non-textual data
Elements of all other Physical Types are non-textual. The length of non-textual data is determined by the Physical Type of the element. For non-textual data, the Data Element Separation property of the parent complex type or group does not determine the length, unless Data Element Separation is Use Data Pattern. See "MRM TDS format: Data pattern separation types" on page 2195 for more information.

The following table describes how the length of data is determined for each Physical Type.

Physical Type	Determination of Length
Text TLOG Specific	The Data Element Separation of the parent complex type or group determines how the length of the data is determined.
External Decimal Integer Packed Decimal Float Time Seconds Time Milliseconds	Uses the value of the Length property of the element. If Physical Type is Time Seconds, the Length property is set to 4. If Physical Type is Time Milliseconds, the Length property is set to 8. In neither case can this value be changed.
Length Encoded String 1 Length Encoded String 2	Uses the encoded length value in the data.
Null Terminated String	Uses the null terminator at the end of the data.
Binary	Uses the value of the Length Reference or Length property of the element.

MRM TDS format: Data element separation
Data element separation defines how a TDS message is to be parsed.

Data element separation defines which method of identifying data elements is to be used and how the data elements are constructed. The different methods vary from full flexibility to fixed format, depending on how they are defined.

The four main types of data element separation are:

Fixed-length types
Fixed-length types are dependent on each element having a length. See "MRM TDS format: Fixed-length separation types" on page 2187.

Tagged separation types
Tagged separation types are dependent on each element having tag prefix to identify it. See "MRM TDS format: Tagged separation types" on page 2188.

Delimited separation types
Delimited separation types use delimiters to identify the end of one data elements and the beginning of the next. See "MRM TDS format: Delimited separation types" on page 2191.

Data pattern types
Data pattern types use a regular expression to identify each element. See “MRM TDS format: Data pattern separation types” on page 2195.

There is a fifth category, which is different from the four described earlier in this topic:

Undefined separation types
Undefined separation types contain no data elements. They are applicable to embedded messages only. They use none of the TDS type-specific parameters other than `Data Element Separation`. See “Message Sets: Multipart messages” on page 2175.

MRM TDS format: Fixed-length separation types
For fixed-length separation types, each data value is a fixed length.

For fixed-length data element separation types, all textual elements have a length or length reference, and are padded out to their full length in the bit stream. No tags or delimiters are used, and each data value directly follows the preceding data value.

For example:

```
data1data200data30
```

The first element is length 5, the second is length 7, and the third is length 6. The padding character is "`0`".

For non-textual elements, the length is determined by the `Physical Type` of the element. See “MRM TDS format: Determining the length of simple data values” on page 2186.

Fixed-length type

In fixed-length type, all textual elements must have a length or length reference, and must be written out to that full length. The elements must be presented in the correct order, and all elements must be written in the bit stream. This includes all repeats of any repeating element (that is, the `Maximum Occurrences` must be written out for each element).

For non-textual elements, the length is determined by the `Physical Type` of the element. See “MRM TDS format: Determining the length of simple data values” on page 2186.

For example:

```
data10data2data2data2data300
```

The first element is length 6, the second is length 5, and repeats three times, and the third element is length 7. The padding character is "`0`".

Applicable parameters

The main parameters for this format are the `Length` or `Length Reference` of the element. All fields must be padded out to their full length for the bit stream to be correctly specified to the parser.

Tag and delimiter parameters are ignored. Group indicators and terminators are observed, because they are of fixed length.

Default values are required for each field that might not be set, because then every field can be produced as output, even if it is not filled with data from the message.

Fixed-length AL3 type (Deprecated)

This separation type has been deprecated. ACORD AL3 support will be provided by a different method in a future release, at which time this separation type will be removed from service.

Fixed-length AL3 types are similar to fixed-length types, but follow extra rules that are specified by the ACORD AL3 format regarding truncation and missing elements. If elements are missing from the end of an AL3 type, they can be truncated. They cannot be omitted from the middle of a bit stream. If a field is missing from the middle of the bit stream, that field is produced for output as the appropriate length string of the "?" character.

Applicable parameters

The main parameters for this format are the `Length` or `Length Reference` of the element. All fields must be padded out to their full length for the bit stream to be correctly specified to a parser.

Tag and delimiter parameters are ignored. `Group Indicators` and `Terminators` are observed, because they are of fixed length.

MRM TDS format: Tagged separation types

For tagged separation types, each data value is preceded by a tag that is specified as an element property.

The `Tag Data Separator`, or specific `Length of Tag` parameter is used to determine where the tag ends and the data starts. Different methods are used by each separation type to determine the end of the data.

After considering these two parameters, this topic describes the following supported tagged separation types:

- "Tagged Delimited separation" on page 2188
- "Tagged Fixed Length separation" on page 2189
- "Tagged Encoded Length separation" on page 2190

Tagged separation is a flexible format. The elements do not have to occur in a specific order. They do not all need to be present, and can be absent from any point in the message.

`Tag Data Separator` and `Tag Lengths`

Either `Tag Data Separator` and `Length of Tag` are used by all tagged separation types. But only one of these parameters can be set at the same time.

The point at which a tag ends and data starts can be determined by one of two methods. If the `Tag Data Separator` is set, then this character indicates where the data ends. For example, the string might be:

```
tag1:data1
```

where `Tag Data Separator` is :

However if the `Tag Data Separator` is not set and the `Length of Tag` field *is* set, then the tag is the specified length, and is immediately followed by the data. No separating character is required. For example, the string might be:

```
tag1data1
```

where `Length of Tag` is 4

Tagged Delimited separation

Tagged Delimited separation is a flexible format. Elements are separated by a predefined delimiter. The textual elements are not of specific lengths. For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 2186.

Applicable parameters

These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Delimiter` separates the data elements within a group or complex type.
- `Tag` for each element, indicates the tag needed to precede the data in that field.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples

If `Tag Data Separator` is set to :

```
{tag1:data1*tag2222222:data2*tag333:data3}
```

where:

- `Group Indicator` is {.
- `Group Terminator` is }.
- `Delimiter` is *.
- Tag defined for each element, is tag1 (for data1), tag2222222 (for data2), and tag333 (for data3).

or, for example, if `Length of Tag` is set to 5

```
{tag11data1*tag22data2*tag33data3}
```

where parameters are as above, except:

- Tag, defined for each element (fixed at five characters), is tag11 (for data1), tag22 (for data2), and tag33 (for data3).

Tagged Fixed Length separation

Although Tagged Fixed Length separation is a flexible format, the data must be a specific length. This means that a delimiter is not needed to determine the end of each element.

Applicable parameters

These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- Tag for each element, indicates the tag needed to precede the data in that field.
- For each textual element, `Length` or `Length Reference` indicates the length of the data (this value does *not* include the length of the tag). For non-textual elements, the length is determined by the `Physical Type` of the element. See “MRM TDS format: Determining the length of simple data values” on page 2186.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples

If `Tag Data Separator` is set to :

```
{tag1:data1tag22222222:data2000tag333:data300}
```

where:

- `Group Indicator` is {.
- `Group Terminator` is }.
- `Delimiter` is *.
- Tag, defined for each element, is tag1 (for data1), tag22222222 (for data2000), and tag333 (for data300).
- `Length`, defined for each element, is 5 (data1), 8 (data2000), and 7 (data300).

or, for example, if `Length of Tag` is set to 5

```
{tag11data1tag22data2000tag33data300}
```

where parameters are as above, except:

- Tag, defined for each element (fixed at five characters), is tag11 (data1), tag22 (data2000), and tag33 (data300).

Tagged Encoded Length separation

This method has both a tag and a length field before the data. The length field indicates to the parser the length of the data following it.

The length of this length field is itself defined in the `Length of Encoded Length` parameter. Extra lengths to be added in this, such as the length of the field itself, is set in the `Extra Chars in Encoded Length` parameter.

Only textual elements and elements with a Binary logical and physical type are supported within a Tagged Encoded Length separation.

These examples show how the values set in these parameters are applied:

- `tagA007dataAAAtagB006dataBBtagC009dataCCCCC`

  If `Length of Tag` is 4, `Length of Encoded Length` is 3, `Extra Chars in Encoded Length` is 0, then in this bit stream, `TagA` is followed by the three character long length field. This indicates that the following data (`dataAAA`) is seven characters long. The next field, `tagB` is then considered, and so on.

- `tagA012dataAAAAAtagB010dataBBBtagC016dataCCCCCCCCC`

  If `Length of Tag` is 4, `Length of Encoded Length` is 3, `Extra Chars in Encoded Length` is 3, then in this bit stream, `TagA` is followed by the three-character length field. This indicates that the following data, plus extra characters, is 12 characters long: length of the length field (3) + length of data (9) = 12. Therefore the length of the actual data is only 12-3 = 9. The next field, `tagB` is then considered, and so on. In each case the length given in the bit stream is 3 greater than the actual length of the data.

Applicable parameters

These parameters are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Tag` for each element, indicates the tag needed to precede the data in that field.
- `Length of Encoded Length` indicates the length of the length field in the bit stream.
- `Extra Chars in Encoded Length` indicates how many extra characters should be included in calculating the value for the length field in the bit stream.
- Either `Tag Data Separator` or `Tag Length` as described earlier in this topic.

Examples

If `Tag Data Separator` is set to :

```
{tag1111:008data1tag222222222:010data2AAtag3333:009data3A}
```

where:

- `Group Indicator` is {
- `Group Terminator` is }
- `Length of Encoded Length` is 3
- `Extra Chars in Encoded Length` is 3
- Tag, defined for each element, is tag1111, tag222222222, tag3333, and so on

or, for example, if `Length of Tag` is set to 5

```
{tag11008data1tag22010data2AAtag33009data3A}
```

where parameters are as above, except:

- Tag, defined for each element (fixed at five characters), is tag11, tag22, tag33, and so on

MRM TDS format: Delimited separation types
For delimited separation types, a delimiter is used to separate data fields, but there are no tags present. The data fields must be given in the correct order in the bit stream and elements cannot be omitted from the middle of the bit stream.

The All Elements Delimited separation type means that data fields are delimited by a pre-specified character or string. In this example, four data fields are separated by an asterisk (*) delimiter:

```
data1*data2*data3*data4
```

Delimited separation types are restrictive in the ordering and presence of elements:

- The elements must be given in the order specified.
- No element can be omitted in the middle of a group or complex type, because the parser cannot determine this from the resulting bit stream.
- Elements can sometimes be absent from the end of a complex type or group.

After considering "Delimiter suppression and truncation rules" on page 2191, this topic describes the following delimited separation types:

- "All Elements Delimited" on page 2192
- "Variable Length Elements Delimited" on page 2193

Delimiter suppression and truncation rules

- Elements cannot be omitted from the middle of a group or complex type. An absent element results in the inclusion of a zero-length string.

  For example, with all elements present, the string might be:

  ```
  data1*data2*data3*data4
  ```

  where `Delimiter` is *

  If `data2` is missing, the string would read:

  ```
  data1**data3*data4
  ```

- It is possible to suppress the delimiters at the end of a string for absent elements. The `Suppress Absent Element Delimiter` property determines whether this is done. If this property is set to End of Type, this can be done (with one exception, shown later in this section).

  In this case, for the example with `data3` and `data4` missing, the string would read:

  ```
  data1*data2
  ```

  That is, the delimiters have been suppressed from the end of this group or complex type.

- If the Suppress Absent Element Delimiter property is set to Never, delimiter suppression never takes place. The string would read:

  ```
  data1*data2**
  ```

  That is, the delimiters must be present to indicate absent (zero-length) elements.

  An exception to the above rule occurs in the case where the same delimiters are used at multiple levels in the model.

  For example, you have a complex type or group with delimiter * and this contains an element of another complex type (indicated by the element3 prefix on data fields in the following example), which also has delimiter *. If both types use a delimited separation type, with all elements present, you might have:

  ```
  data1*data2*element3Data1*element3Data2*element3Data3*data4
  ```

  If element3Data2 and element3Data3 are missing, and the delimiters are suppressed, it is not possible for the parser to determine which elements are missing.

  Therefore, in this case, you must override the Suppress Absent Element Delimiter property, and write out all the delimiters to clearly define the message to the parser. Therefore, the string must be:

  ```
  data1*data2*element3Data1***data4
  ```

  This restriction also applies where Group Indicators and Group Terminators use the same character strings as delimiters; otherwise, the bit stream is not clear to the parser.

All Elements Delimited

In an All Elements Delimited separation type, all elements are separated by a delimiter; for example:

```
data1*data2*data3*data4*data5
```

where Delimiter is *.

An All Elements Delimited separation type does not use tags or their associated parameters.

For textual elements, the length is determined by the delimiter, and the Length property is ignored unless the Observe Element Length property is set.

For non-textual elements, the length is determined by the Physical Type of the element. See “MRM TDS format: Determining the length of simple data values” on page 2186.

Applicable properties

These properties are used:

- Group Indicator indicates the start of a group or complex type.
- Group Terminator indicates the end of a group or complex type.
- Delimiter indicates the separator between the data elements within a group or complex type.
- Suppress Absent Element Delimiters indicates whether delimiter suppression is permitted.

For example:

```
{data1*data22222*data3}
```

where:

- Group Indicator is {
- Group Terminator is }
- Delimiter is *

Repeating element rules

If an element must be repeated when the separation type is All Elements Delimited, the `Repeating Element Delimiter` (RED), is used to separate the repeated elements.

For example if `data2` repeats five times:

```
data1*data2:data2:data2:data2:data2*data3*data4
```

where:

- `Delimiter` is *
- `Repeating Element Delimiter` is:

If the `Suppress Absent Element Delimiters` property is set to End of Type, you can use delimiter suppression. Therefore, if only the first `data2` element was present in the previous example, the bit stream reads:

```
data1*data2*data3*data4
```

However, if the `Suppress Absent Element Delimiters` property is set to Never, the bit stream reads:

```
data1*data2::::*data3*data4
```

If `Delimiter` and RED match, two delimiters are output to indicate that the repeat is ending. Therefore, if the delimiter and RED are *, the bit stream reads:

```
data1*data2**data3*data4
```

Variable Length Elements Delimited

In a complex type with Variable Length Elements Delimited separation, some elements are determined by their length, and other elements are delimited. This combination of a delimited and a fixed-length format follows rules that are associated with both formats. Lengths can be given and used, but they are not mandatory.

- If a length is present for a textual element, it is used, and a delimiter is not needed to terminate that element. The element must be padded to the correct length, and cannot exceed that length.
- If no length is given for a textual element, the delimiter is required.
- For non-textual elements, the length is determined by the `Physical Type` of the element. See "MRM TDS format: Determining the length of simple data values" on page 2186.

A complex type with Variable Length Elements Delimited separation that contains only variable length elements resembles a complex type with All Elements Delimited separation. If it contains only fixed-length elements, it resembles a Fixed Length type.

For example:

```
data1*data2*data3*data4000data5
```

where:

- `Delimiter` is *
- `data4` has a length of 8

Applicable properties

The following properties are used:

- `Group Indicator` indicates the start of a group or complex type.
- `Group Terminator` indicates the end of a group or complex type.
- `Delimiter` indicates the separator between the data elements within a group or complex type.

- `Suppress Absent Element Delimiters` indicates whether delimiter suppression is permitted.
- (Optionally) `Length` or `Length Reference` indicates the length of a textual element. If a textual element has a length, this length is used. Because the length of this element is known, it is not necessary to output a delimiter after it. If the length is not known, a delimiter is required. A delimiter is never required for a non-textual element.

In this example, the fourth field (containing `data4`) is of fixed length 8 and its padding character is `0`:

```
{data1*data22222*data3*data4000data5}
```

where:

- `Group Indicator` is {
- `Group Terminator` is }
- `Delimiter` is *

Repeating element rules

The action of a repeating element in a Variable Length Elements Delimited environment is dependent on the minimum and maximum number of repeats and whether the element has a length.

Delimited Element Repeating

If a delimited element (that is, an element with no length) is repeated, a `Repeating Element Delimiter` (RED) is required and the rules for All Elements Delimited are followed. A delimiter is therefore required after the last repeat. Delimiter suppression of this repeat can also occur.

For example, if `data2` is repeating:

```
data1*data2:data2:data2:data2:data2:data2*data3*data4000data5
```

where:

- `Delimiter` is *
- `Repeating Element Delimiter` is :
- `data4` has a fixed length of 8

If the `Suppress Absent Element Delimiters` field is set to End of Type, you can use delimiter suppression.

If in the above example only the first `data2` is present:

```
data1*data2*data3*data4000data5
```

However, if `Suppress Absent Element Delimiters` is set to Never, the bit stream reads:

```
data1*data2:::::*data3*data4000data5
```

If the delimiter and RED match, two delimiters are output to indicate that the repeat is ending. So if the delimiter and RED are both *, the bit stream reads:

```
data1*data2**data3*data4
```

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

Fixed Length Element Repeating

If an element with a defined length (a fixed-length element) is repeating and the minimum occurrences is not the same as maximum occurrences, an RED is not required, but a delimiter *is* required after the last repeat. Delimiter suppression of this repeat can occur.

For example, if data4 (with a fixed length of 8) is repeating, and its minimum occurrences is 2, maximum occurrences is 4:

```
data1*data2*data3*data400data400data400data400*data5
```

where Delimiter is *

Or, if there are only two occurrences of data4:

```
data1*data2*data3*data4000data4000*data5
```

If an element with a defined length (a fixed-length element) is repeated, and the minimum occurrences is the same as maximum occurrences, an RED is not required. A delimiter is also not required after the last repeat. Truncation of this repeat cannot occur and all elements need to be present.

For example, if data4 (with a fixed length of 8) repeats four times:

```
data1*data2*data3*data4000data4000data4000data4000data5
```

where Delimiter is *

Or, if there are only two occurrences of data4:

```
data1*data2*data3*data4000data4000000000000000000data5
```

This also applies for a non-fixed length complex type or group inside a Variable Length Elements Delimited environment.

If a complex type has Variable Length Elements Delimited separation, a delimiter is always output between an included ('child') complex element and the next element even if the separation of the 'child' complex element is Fixed Length. On input, the parser accepts the bit stream with or without such a delimiter.

MRM TDS format: Data pattern separation types
For a data pattern separation type, each data value is matched with a regular expression that is specified as a property of each element.

The length of both textual and non-textual data is determined by the Data Pattern property of the element. If the Physical Type of the element is Length Encoded String 1 or Length Encoded String 2, the regular expression must match both the encoded length and the following data. The length in the encoded length must be consistent with the length matched by the regular expression. If the Physical Type of the element is Null Terminated String, the regular expression must match both the data and the following null terminator.

The Data Pattern separation type uses a regular expression that is specified for each element to match the data. The parser matches the data with the regular expression in the Data Pattern property for that element. TDS parsing in the MRM parser uses the regular expression in Data Pattern to determine the length of the element, whether it is repeating, and whether it is present in the bit stream.

No delimiters or tags, other than those coded as part of the regular expression pattern, are used in the bit stream. See Message Sets: Using regular expressions to parse data elements for an explanation of pattern matching.

For example, if the first three Data Pattern properties are, respectively:

- [A-Z]{1,3}
- [0-9]+
- [a-z]*

and the message data is:

```
DT31758934information for you
```

Then, in this example:

- First data element = DT
- Second data element = 31758934
- Third data element = information

The first data pattern means "from one to three characters in the range A to Z", the second means "one or more characters in the range 0 to 9", and the third means "zero or more characters in the range a to z". Note how each element's data was terminated by the first character that did not match the element's Data Pattern.

If the TDS message that is being parsed is encoded in a single-byte code page, the Data Pattern property can include hexadecimal values. A hexidecimal value is specified as \xNN, where N is a hexadecimal digit in the range 0 to F. Note, however, that the value \x00 is not valid.

Performance issues

The parsing required in Data Pattern separation type is the slowest of all the different separation types because of its complexity.

Therefore, use Data Pattern separation type only when no other separation type models the message. Do not use it, for example, when you can use Fixed Length separation type.

Applicable parameters

Only one parameter is used:

Data Pattern for each element, indicates the regular expression that is used for string matching.

MRM TDS format: Message model integrity
When you use the TDS wire format, you must conform to a number of rules that apply to the setting of values of properties. This is necessary to avoid any discrepancies when processing a message within the specified model.

Rules of TDS physical format properties

Restrictions to message formats are checked. These restrictions follow the rules specified in Message Sets: TDS message model integrity. Most rules are applied for at least one of these reasons:

Rules for message definition
Some rules are necessary for the message to be defined.

For example, in a Fixed Length separation type all elements must have some length defined, either directly or by using a Length Reference. Without this information, it is impossible to tell in the message bit stream where one data element ends and the next starts.

Rules for nesting
Nesting rules relate to which separation types can be nested inside each other.

Such rules are applied when an element of a complex type is present inside another complex type. An example is that it is not possible to have a Tagged Delimited separation type inside a Fixed Length type. Because a Tagged Delimited separation type is of variable length, the parent Fixed Length type would be unable to tell where that particular element ended, as there would be no length provided. Therefore the message could not be processed.

Rules linking to the logical model
There are also rules linking TDS to the logical model.

These rules occur where a group composition or group content validation cannot be used with a particular separation type. Again this is for message integrity. For example, a separation type of All Elements Delimited cannot have a group composition of Open, as there is no information as to what the extra elements represent and where they are in the bit stream.

MRM TDS format: NULL handling
NULL handling dictates the way in which the MRM parser for TDS messages handles elements that have been set to Null.

Null handling takes place only if the logical Nillable property of the element is set. The rules for whether nulls are permitted are described in Message Sets: TDS Null handling options.

Null properties

The element properties `Encoding Null` and `Encoding Null Value` control how null handling is represented for individual elements.

You can select the `Encoding Null` property from the enumerated values NULLPadFill, NULLLogicalValue, NULLLiteralValue, and NULLLiteralFill. The use of the `Encoding Null Value` property is dependent on the value that you select for the `Encoding Null` property.

NULL values are not defined for schema xsd:hexBinary simple types. The properties `Encoding Null` and `Encoding Null Value` are therefore not set for xsd:hexBinary types.

NULL values for schema `Boolean` simple types are defined at the message set level. The message set property `Boolean Null Representation` specifies the value to be used for Boolean Null representation.

MRM TDS format: Multipart messages
The Tagged/Delimited String Format (TDS) supports both the Message Identity technique and the Message Path technique of identifying embedded messages within a multipart message.

The SWIFT, X12, and EDIFACT messaging standards can all be modeled by using the Message Identity technique.

MRM TDS format: Data conversion
TDS string data is subject only to CCSID conversion.

All TDS message data apart from binary types are handled as strings. All string data is therefore subject to CCSID conversion only. This includes the special characters used as delimiters, data separators, and so on \.

MRM TDS format: Relationship to the logical model
TDS separation types and logical model properties have some restrictions, such as group composition and group content validation.

The rules that govern these options are explained in Message Sets: Restrictions for nesting complex types.

These rules exist to ensure the integrity of the message. A combination of separation type and group composition or group content validation must not lead to a message that is unclear to a TDS parser.

Default values

In TDS, *Default* values are only observed by fixed-length elements:

Separation Type	**Use of `Default` values**
Tagged Delimited Tagged Fixed Length Tagged Encoded Length All Elements Delimited Data Pattern	`Default` values are never observed.
Fixed Length Fixed Length AL3	`Default` values are observed on output by all elements. An absent element that has no `Default` value defined, causes an error on writing.

Separation Type	Use of `Default` values
Variable Length Elements Delimited	`Default` values are only observed by fixed-length elements on output. Absent fixed-length values must have a `Default` value available to them. An absent element that has no `Default` value defined, causes an error on writing.

Simple types - lists and unions

Lists and unions are XML-specific concepts. An element or attribute of a simple type that is a list or a union causes a task list warning if a TDS physical format is present in the message set. The user can choose whether to make this an error, warning, or information by editing the Validation preferences. If a dictionary is generated from the message set, and an attempt is made to parse a TDS message defined to contain such elements or attributes, a runtime error occurs.

`Min Occurs` and `Max Occurs`

The logical properties `Min Occurs` and `Max Occurs` specify the permitted number of occurrences of an element or group in a message. They are used when parsing and writing messages, and when validating the content of a message.

When parsing and writing, the exact interpretation of these properties depends on the `Data Element Separation` property of the parent complex type or group as shown in the following table.

However, this behavior is overridden if the TDS `Repeat Reference` property is set, which indicates that the number of occurrences is given instead by an integer element that occurs earlier in the message. See “Repeat reference” on page 2200 for more information.

When validating, `Min Occurs` and `Max Occurs` are both used to check that the content of the message tree matches the model.

Separation type	Interpretation of `Min Occurs` and `Max Occurs`
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	`Min Occurs` and `Max Occurs` are effectively ignored when parsing and writing. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree. • A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is allowed. • Optional occurrence (`Min Occurs` = 0) is allowed. • Always absent (`Max Occurs` = 0) is allowed. • An unbounded number of occurrences (`Max Occurs` = -1) is allowed.

Separation type	Interpretation of Min Occurs and Max Occurs
All Elements Delimited	Max Occurs is used only when parsing and writing, with the element's Repeating Element Delimiter property, and the parent type's Suppress Absent Element Delimiters property. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed if Suppress Absent Element Delimiters is set to End of Type. • If the Delimiter is different from the Repeating Element Delimiter, the Delimiter signifies the end of the occurrences. • If the Delimiter is the same as the Repeating Element Delimiter, an empty repeat signifies the end of the occurrences. • In both these cases, Max Occurs is the maximum number of repeats that are expected. If Suppress Absent Element Delimiters is Never, all occurrences are expected when parsing, and produced when writing, although parsing accepts elements being absent. Optional occurrence (Min Occurs = 0) is ignored and a delimiter is still expected when parsing, and produced when writing. Always absent (Max Occurs = 0) is allowed. No delimiter is expected when parsing, nor output when writing. An unbounded number of occurrences (Max Occurs = -1) is only allowed if the Repeating Element Delimiter is different from the Delimiter. The repeats must be terminated by the delimiter, or a containing group's Group Terminator or Delimiter, or by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree.
Fixed Length Fixed Length AL3	Max Occurs is used only when parsing and writing. In general, Max Occurs occurrences are expected when parsing, and Max Occurs occurrences are produced when writing; default values are used for missing elements, and any excess elements are discarded. A varying number of occurrences (Min Occurs <> Max Occurs) is ignored, Max Occurs is assumed. Optional occurrence (Min Occurs = 0) is ignored, Max Occurs is assumed. Always absent (Max Occurs = 0) is allowed. Fixed Length only. An unbounded number of occurrences (Max Occurs = -1) is allowed if the element or group is the last child in its parent group, and the group is terminated by a Group Terminator or a containing group's Group Terminator or Delimiter or by the end of the message bit stream. On writing, the writer outputs all occurrences in the message tree, if this number is less than Min Occurs, additional default values are written.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed with two differences. 1. A varying number of occurrences (Min Occurs <> Max Occurs) is allowed, the end of the occurrences being signified by an extra delimiter. 2. An unbounded number of occurrences (Max Occurs = -1) is allowed, the end of the occurrences being signified by an extra delimiter. On writing, the writer outputs all occurrences in the message tree, followed by an extra delimiter. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited above are followed.

Separation type	Interpretation of `Min Occurs` and `Max Occurs`
Data Pattern	`Min Occurs` and `Max Occurs` are effectively ignored when parsing and writing. When parsing, the pattern is matched as many times as possible. When writing, the writer outputs all occurrences in the message tree. Note that on parsing, if the data pattern permits a zero length match, and a zero length match occurs, an element is added to the message tree, and the matching terminates to prevent an infinite loop. A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is allowed. Optional occurrence (`Min Occurs` = 0) is allowed. Always absent (`Max Occurs` = 0) is allowed. An unbounded number of occurrences (`Max Occurs` = -1) is allowed.

`Repeat reference`

The TDS property `Repeat reference` specifies a field that holds the number of repeats of an object (Element or Group) within a message. The field that holds the number of repeats must be within the message before the object that it refers to.

From a parsing perspective, the `Repeat reference` property replaces the role of the `minOccurs` and `maxOccurs` properties.

If a value for the `Repeat reference` property is specified for an object, values that are specified for `minOccurs` and `maxOccurs` are ignored when parsing and writing. However, values that are specified for `minOccurs` and `maxOccurs` are used by logical validation.

When parsing and writing, the exact interpretation of the `Repeat reference` property depends on the `Data Element Separation` property of the parent complex type or group as shown in the following table.

Separation type	Interpretation of `Repeat reference`
Tagged Delimited Tagged Fixed Length Tagged Encoded Length	`Repeat reference` is effectively ignored when parsing and writing. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.
All Elements Delimited	`Repeat reference` is used when parsing and writing, with the element's `Repeating Element Delimiter` property, and the parent type's `Suppress Absent Element Delimiters` property. A `Repeat reference` is allowed only if the parent complex type or group has `Suppress Absent Element Delimiters` set to Never. All `Repeat reference` occurrences are expected when parsing, and produced when writing. However, parsing accepts elements being absent. `Repeat reference` = 0 is allowed. No delimiter is expected when parsing, nor produced when writing.
Fixed Length Fixed Length AL3	`Repeat reference` is used when parsing and writing. `Repeat reference` occurrences are expected when parsing, and are produced when writing, with default values used for missing elements. `Repeat reference` = 0 is allowed.
Variable Length Elements Delimited	For fixed length simple elements, the rules for Fixed Length separation above are followed. For variable length simple elements, all complex elements and groups, the rules for All Elements Delimited that are listed above are followed.

Separation type	Interpretation of `Repeat reference`
Data Pattern	`Repeat reference` is effectively ignored when parsing and writing. When parsing, the pattern is matched as many times as possible. When writing, the writer outputs all occurrences in the message tree. Note that, on parsing, if the data pattern permits a zero length match, and a zero length match occurs, an element is added to the message tree, and the matching terminates to prevent an infinite loop.

MRM XML physical format

The MRM XML physical format describes the physical representation of an XML message for use by the MRM parser.

Tip: In Version 8, when modeling and parsing XML data, use message model schema with the XMLNSC or SOAP domains, instead of message sets and the MRM domain.

An XML wire format describes the physical representation of a message that is written according to the standards given in the W3C Extensible Markup Language (XML) specification. The wire format defines information that is used to parse or write XML messages in a runtime environment such as an integration node. XML versions 1.0 and 1.1 are both supported.

You can add more than one XML physical format to a message set, but within that message set, each physical format must have a unique name. The default name for an XML wire format is XML1. The physical format name identifies the definitions that are to be used by the integration node at run time.

After adding an XML physical format, all XML properties for all existing objects in the message set are set to default values. Therefore, immediately after adding the format and deploying the message set to a runtime environment, you can process XML messages by using MRM features.

You can configure XML properties for the message set, and for objects within the message set. Objects that can have XML properties are messages, elements, and attributes. For example, a message object can be customized to define a specific DTD declaration on output; an element can have a tag name assigned to it which is different from its element name in the model.

Adding an XML wire format to a message set allows you to both process input messages, and to construct output messages in this format. You can also transform messages between XML and either CWF or TDS.

XML messages are, by their nature, self-describing: each piece of data is prefixed by a tag name or an attribute name. Therefore, it is possible for an XML message instance to contain elements that are not in the definition for that message.

- If such an element exists in the message set, the model objects for that element are used in parsing or writing the message.
- If the element does not exist in the message set, it is treated as a self-defining element, and its data type is set to string.

Although it is possible to define an XML message 'by hand', using the Message Definition Editor, IBM App Connect Enterprise also provides importers for both XML Schema and DTD, and these are often quicker and easier than manual definition.

MRM XML physical format: Message model integrity

When you save a message definition file, the definitions that it contains are checked to ensure that they make sense and provide sufficient information about the message. This action is called *model validation*.

For XML, these checks mostly concern the uniqueness and validity of XML names in global elements and attributes, and also for elements and attributes within complex types or groups.

Tests other than these tend to be both simple and obvious so that, for example, the message set property `First Day of Week` must be the name of a day in the week.

MRM XML physical format: NULL handling
The purpose of null handling is to specify how messages deal with null values; that is, the absence of a meaningful value for an element.

Null properties for the MRM XML physical format are set for the message set only, and apply to all the defined objects within the message set, by using the four properties `Encoding Null Num`, `Encoding Null Non-Num`, `Encoding Null Num Val` and `Encoding Null Non-Num Val`.

Null handling takes place only if the logical Nillable property of the element is set.

The purpose of these parameters is to specify how messages deal with null values. In an XML message there are several options. Most obviously an element could omit a value, for example:

```
<element1></element1>
```

Or, the element could include a distinctive value that means that no real value is present, for example.

```
<element1>null</element1>
```

Or, the element could follow XML Schema instance rules:

```
<element1 xsi:nil="true"/>
```

The properties `Encoding Null Num` and `Encoding Null Non-Num` specify the style of null handling, for example, that null is represented by an empty element.

The properties `Encoding Null Num Val` and `Encoding Null Non-Num Val` provide a value (if needed) to represent a null value. For an element of type string, this might be `null` or `unspecified` while for a number it might be `0` or `0.0`.

MRM XML physical format: Multipart messages
Identify embedded messages by using either a Message Identity or a Message Path.

If you are using the MRM XML physical format, an embedded message can be identified in any of the following ways:

- Message Identity

  See “Message Sets: Identifying an embedded message by using a Message Identity” on page 2176.
- Message Path

  See “Message Sets: Identifying an embedded message by using a Message Path” on page 2178.
- Automatic

  The MRM parser identifies the message by matching the next XML tag in the bit stream against the XML Name of a message definition.

If you choose the Message Identity or Message Path technique, the MRM parser still checks that the next XML tag name matches the XML Name of the message that was identified. If the XML Name does not match, an exception is thrown.

Where you have defined the embedded message in a different message set, you must use a Message Set Identity element or attribute value to specify the target message set. Note that the message sets within which the root and subsequent embedded messages are defined must be consistent in their use of the 'Use Namespace' property of the message set. That is, embedded messages that are defined in a namespace-aware message set and that are contained within a parent message that is defined in a message set that is not namespace-aware, are not supported. Similarly, embedded messages that are defined in a message set that is not namespace-aware and that are contained within a parent message that is defined in a namespace-aware message set, are not supported.

If the embedded message definition is a complex type, the message definition contains a complex element based on that complex type. This complex element has its own tag, which appears in the bit stream before the tag for the embedded message. If you want to avoid this extra tag, you can create

the embedded message definition from a group, and insert the group at the appropriate position in the message model.

Tip: Note that the root tag property of an embedded message is not applicable.

MRM XML physical format: Relationship to the logical model
The MRM XML physical format generally respects all the settings in the logical model, but shares certain restrictions in common with the other physical formats.

These restrictions are documented in MRM restrictions.

Default values

The MRM XML physical format ignores default and fixed values on elements and attributes. If validation is enabled in IBM App Connect Enterprise, this can lead to unexpected validation errors for missing elements, even though they have default or fixed values.

Simple types - unions and lists

The XML properties of an element or attribute of a simple type that is a union or list vary depending on the members of the union or the itemType of the list. If the union or list includes a dateTime type (or other date/time related type) the `Date Format` field is displayed. If the union includes a binary type, the `Encoding` field is displayed.

Min Occurs and Max Occurs

The logical properties `Min Occurs` and `Max Occurs` specify the permitted number of occurrences of an element or group in a message. They are used when validating the content of a message.

When parsing and writing, using the MRM XML physical format, `Min Occurs` and `Max Occurs` are effectively ignored. When parsing, the number of occurrences is identified by the tags in the message. When writing, the writer outputs all occurrences in the message tree.

- A varying number of occurrences (`Min Occurs` <> `Max Occurs`) is allowed.
- Optional occurrence (`Min Occurs` = 0) is allowed.
- Always absent (`Max Occurs` = 0) is allowed
- An unbounded number of occurrences (`Max Occurs` = -1) is allowed.

When validating, `Min Occurs` and `Max Occurs` are both used to check that the content of the message tree matches the model.

MRM XML physical format: Handling xsi:type attributes
The prefix "xsi" is the namespace prefix used by convention for the XML Schema instance namespace. XML documents can contain elements that have an xsi:type attribute. This behavior provides an explicit data type for the element.

The MRM XML parser in sensitive to xsi:type attributes in the XML document. It modifies the data type of the element accordingly and adds the xsi:type attribute into the message tree.

The MRM XML writer is sensitive to xsi:type attributes in the message tree. It produces xsi:type attributes according to XML Wire Format message set property **Output policy for xsi:type attributes**. For example, xsi:type attributes can be removed, output on all elements or output according to rules specified in the SOAP standard.

If validation is enabled for an IBM App Connect Enterprise message flow, the validation logic is sensitive to xsi:type attributes and uses them to modify the validation of the element. It also validates the values of xsi:type attributes by using the rules described in XML Schema Part 1: Structures on the World Wide Web Consortium (W3C) website.

There are several important points to remember when parsing and writing XML documents that contain xsi:type attributes.

- In order to detect and use xsi:type attributes, the message set must be namespace-enabled. To make a message set namespace-enabled, check the message set property **Use namespaces**.
- If the value of the xsi:type attribute contains a namespace prefix, the prefix is expanded into a fully qualified URI by the MRM XML parser. If the same xsi:type attribute is produced later by the MRM XML writer, the same prefix is not automatically used for the value. You can control the prefixes used on output by using the **Namespace settings** list in the XML Wire Format message set properties. If no prefix is supplied, the XML writer assigns a default prefix.
- If the xsi:type attribute of an element does not resolve to a type in the model, the behavior depends on whether MRM validation is enabled. If not validating, the MRM assumes that the type of the element is that declared in the model, and continue. If validating, a validation exception occurs.
- If MRM validation is enabled, any required xsi:type attributes must be present in the message tree at the point when validation is performed. An xsi:type attribute is required when its value is different from the data type of the element as defined in the message model (this most commonly occurs when using XML Schema type derivation).
  - If validation is being performed on an input message, the MRM XML parser ensures that xsi:type attributes appear in the message tree, as described above.
  - If validation is being performed on an output message, you must ensure that the correct xsi:type attributes appear in the message tree. Ensure that any required xsi:type attributes are copied from input message tree to output message tree, or are explicitly created in the output message tree.
- If you are using simple types that are xsd:unions, an xsi:type attribute can be used to direct the MRM XML parser when resolving the union.
- If you have migrated from an earlier version of WebSphere Message Broker that was not sensitive to xsi:type attributes, you might notice some changes of behavior. For example, xsi:type attributes are no longer treated as self-defining attributes, so they appear in the message tree with the name ‘type' instead of ‘@type'. If your message flow logic is sensitive to xsi:type attributes in the message tree, change your message flow to comply with the current behavior.

For more information about xsi:type attributes, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) website.

Message Sets: Generate message dictionaries

A *message dictionary* is data structure that describes all of the messages in a message set in a form suitable for deployment to the MRM parser.

Purpose of a message dictionary

A dictionary describes the logical structure and content of a set of messages, and typically contains one or more physical formats that describe how those messages are serialized in a bit stream. The MRM parser within IBM App Connect Enterprise uses this information to parse an incoming message bit stream into a message tree, or to write a message tree into a physical bit stream.

Contents of a message dictionary

A message dictionary contains the same information as the message set from which it was created, but in a compressed form that the MRM parser within IBM App Connect Enterprise can understand and use. A message dictionary contains all the elements in the message set, the structure of the messages, and all the value constraints. A message dictionary also contains any physical formats that were defined in the message set.

Generating a message dictionary

Before a message dictionary can be used, it must be deployed to IBM App Connect Enterprise. To do this, add the message set to a BAR file, then deploy the BAR file to an integration node. The generation of the message dictionary is performed automatically when a message set is added to a BAR file, if the message set supports the MRM domain.

Before adding a message set to a BAR file, the IBM App Connect Enterprise Toolkit performs a full validation of the message set. If this validation finds any errors, the message set is not added to the BAR file, and therefore no message dictionary is generated.

Dictionary generation report files
Whenever a message dictionary is generated, a user log file is also generated and added to the same BAR file. This file contains informational messages and warnings that relate to the use of the generated dictionary. Always check this file after generating a message dictionary.

Message model integrity

When you create your message model, it is important that it is internally consistent.

To assist with this, whenever you save a message model schema file or message definition file, it is validated as follows:

Logical validation
This validation ensures that the logical model is correct. For message definition files, this involves ensuring that the rules of XML Schema have been correctly followed.

Physical validation
This validation ensures that any physical properties that you have specified for your model have been correctly populated and are consistent. There is a set of checks for each of the MRM domain physical formats - CWF, XML, and TDS - and for DFDL.

After model validation has taken place, any errors or warnings are shown in the task list. Double clicking a task list entry opens the file and positions the editor at the object in error. Organize the task list so that errors are shown before warnings. In this way, errors are not hidden. The task list provides a comprehensive filtering capability if you want to hide low priority warnings, or warnings that you are know about and are comfortable with.

The deployment of a message model in a BAR file is prevented if any errors are present. The presence of warnings alone does not prevent deployment, but high priority warnings must be reviewed because a model that generates such warnings might be incomplete.

Where task list warnings or errors occur, these are listed in the Problems view of the Integration Development perspective. While a majority of these require you to manually investigate and resolve them, a number of warnings and errors that meet specific criteria can be repaired by using a quick fix process.

Generation of model representations

You can use your message model to facilitate application development and to parse and validate messages.

After you have created and populated your message model, you can use it in the following ways:

- At development time to speed up message flow application development, by using the models as the source and target of a map in the Map editor, or by enabling ESQL code assist in the ESQL editor.
- At run time for parsing and validating your messages, by deploying them to an integration node.

Additionally, you can generate your message model in a different form for use by other message flows or other external applications. For example:

- W3C XML Schema, for use by an application building or processing equivalent XML messages.
- Web Services Description Language (WSDL), for a web services client or server application.
- Documentation, to give to programmers or business analysts.

Generating your model in a different form is achieved by using the **Generate** menu actions.

Generation of XML schema
You can generate a schema file from a message model.

XML schema is a standard way of describing complex message models.

You can generate an XML schema file for an individual DFDL schema file in an application or library, an individual message definition file in a message set, or for all message definition files in a message set.

Generating from DFDL

A DFDL schema file is a valid XML schema file, so the generation process consists of removing the DFDL annotations from the DFDL schema. The result is a pure W3C XML Schema file.

Generating from message sets

If any XML physical formats have been defined for the message set, you can select which of these XML wire formats are to be applied.

- If an XML format has been selected, the physical format information is also included.
- If no XML format is selected, the generated schema file contains information about only the logical message model.

You can choose whether 'strict' or 'lax' schema generation is to be performed. This is necessary because the logical extensions to the XML schema model provided by the message definition file cannot be represented in XML schema. So you can choose either to generate a schema with more strict or more lax validation than the equivalent validation performed by the message model parser. The affected model extensions are:

- `Content Validation` = open
- `Content Validation` = open defined
- `Composition` = unordered set

Further information about XML schema

For details about XML schema, see XML Schema Part 0: Primer on the World Wide Web Consortium (W3C) Web site.

Generation of WSDL files

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

Tip: If you need to generate a WSDL document, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see “Enabling message set development” on page 2248.

You can generate a WSDL document for a message model that resides in a message set.

Use message model files with target namespaces when you generate WSDL. If you do not, IBM App Connect Enterprise uses the WSDL target namespace as the default target namespace.

In the main WSDL document, operations are defined in terms of logical messages, which are themselves defined in terms of the elements and types that are defined in these message model files.

WSDL operations are grouped into a logical interface or portType, and are then associated with a binding which defines the physical format of the messages. You can select only one of the following bindings when you generate WSDL:

- SOAP (over JMS)
- SOAP (over HTTP)

A WSDL service definition specifies the endpoint where the service is available. You can elect to have the service, binding, and portType definitions generated as a single file or as separate files. Tools that use WSDL are typically more tolerant of the single-file format. If you select single-file, you can also choose whether the associated XML Schema is generated from the message model as a separate file or in-line.

Considerations for deploying a WSDL file with regular expression definitions

The W3C XML Schema standard specifies how to define regular expressions using the pattern facet of simple type definitions in XML schemas, and WSDL files. Regular expressions can include a quantifier that specifies how many instances of a character, group, or character class must be present in the input, in order for a match to be found.

When the WSDL is deployed to an integration server, it is compiled into a binary file, which the IBM App Connect Enterprise XMLNSC parser uses to validate input messages at run time. The total size of the binary file is the accumulated size of each regular expression, which depends on the quantifier ranges in each expression. A large binary file requires the integration server JVM heap size to be large enough to enable the WSDL file to be successfully deployed.

The following is an example where the quantifier is exceptionally large:

```
<xsd:simpleType name="VeryLargeBoolean">
    <xsd:restriction base="xsd:string">
       <xsd:pattern value="([T|t][R|r][U|u][E|e])|([F|f][A|a][L|l][S|s][E|e]){0,2147483647}" />
    </xsd:restriction>
</xsd:simpleType>
```

The quantifier in this example is {0,2147483647}, which during deployment will generate a binary file that is at least 2 GB in size.

The accumulated size of the regular expressions can cause a very large binary file to be created at deployment time, which might exhaust the integration server JVM heap storage. In the example shown above, the deployment failed with the following sequence of error messages:

```
  BIP4041E: Integration server '<Server name>' received an administration
  request that encountered an exception.

  BIP5044E: A failure occurred while the integration server was preparing
  the XML and DFDL schema files that are contained in the application
  <Application Name>.

  BIP5043E: The XMLNSC compiler ran out of memory during the preprocessing
  of XML Schema files for the XMLNSC domain.
```

If this problem occurs, modify the range of the quantifiers in the regular expressions to generate a smaller number of pattern matching instances, and therefore a smaller binary WSDL file.

Generation of message model documentation

When you have created one or more message models, it can be useful to generate documentation for business analysis and for developers who are involved with the messages.

Message model schema files contain both logical and physical definitions for the message model. The generated documentation describes the logical information only.

The documentation exists as an HTML file.

Modeling different data formats

You can use IBM App Connect Enterprise to create message models for a range of data formats.

About this task

Most message formats are not self-defining, and a parser must have access to a predefined model that describes the message, if it is to parse the message correctly. Even if your messages are self-defining, and do not require modeling, message modeling has advantages. For a full description of the advantages of using message models, see “Why model messages?” on page 2138.

The topics in this section describe how to create a message model for different data formats by using IBM App Connect Enterprise.

- “Working with web services XML data” on page 2208
- “Working with other XML data” on page 2208

- “Working with CSV data” on page 2209
- “Working with other delimited text data” on page 2209
- “Working with custom text or binary data by using DFDL” on page 2209
- “Working with COBOL data” on page 2210
- “Working with C data” on page 2210
- “Working with PL/I data” on page 2210
- “Working with CORBA data” on page 2211
- “Working with SAP data” on page 2211
- “Working with Siebel data” on page 2212
- “Working with PeopleSoft data” on page 2212
- “Working with JD Edwards data” on page 2212
- “Working with database records” on page 2212
- “Working with JSON data” on page 2213
- “Working with MIME data” on page 2213

Working with web services XML data

You can create a message model for your SOAP XML data by using XML Schema.

About this task

SOAP is an XML message format used in web service interactions. SOAP messages are typically sent over HTTP or JMS, but other transport protocols can be used.

For more information about SOAP, see “What is SOAP?” on page 807

You can create a message model for your web services XML data in IBM App Connect Enterprise by using XML Schema; see:

- “Opening an existing XML schema file” on page 2242
- “Creating an XML schema file” on page 2243
- “Deleting an XML schema file” on page 2244
- “Generating HTML documentation from an XML Schema file” on page 2245
- “Working with XML Schema message model objects” on page 2245

Working with other XML data

You can create a message model for your XML data by using XML Schema.

About this task

XML, or EXtensible Markup Language, is a platform-independent data format. For more information about XML messages, see XML constructs

You can create a message model for your XML data in IBM App Connect Enterprise by using XML Schema; see:

- “Opening an existing XML schema file” on page 2242
- “Creating an XML schema file” on page 2243
- “Deleting an XML schema file” on page 2244
- “Generating HTML documentation from an XML Schema file” on page 2245
- “Working with XML Schema message model objects” on page 2245

Working with CSV data

You can create a message model for Comma Separated Value (CSV) data by using DFDL schema.

About this task

The Comma Separated Value (CSV) format is used to exchange data between database applications or spreadsheet applications. Although the CSV format is widely used, a definitive specification has not been formally documented.

For more information about the CSV format, see CSV messaging standard

You can create a message model for your CSV format data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with other delimited text data

You can create a message model for delimited text data by using DFDL schema.

About this task

You can create a message model for your delimited text data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with custom text or binary data by using DFDL

Create a message model for custom text or binary format data by using DFDL schema.

About this task

You can create a message model for your custom text or binary format data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221

- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with COBOL data

You can create a message model for COBOL data by importing a COBOL copybook to create a DFDL schema.

About this task

You can create a message model for your COBOL data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with C data

You can create a message model for C data by importing a C header file to create a DFDL schema.

About this task

You can create a message model for your C data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with PL/I data

You can create a message model for PL/I data by using DFDL schema.

About this task

You can create a message model for your PL/I data in IBM App Connect Enterprise by using DFDL schema; see:

- “Creating a DFDL schema file by using the New Message Model wizard” on page 2223
- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223

- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Working with CORBA data

You can create a message model for data from a CORBA system by using XML Schema.

About this task

The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together.

For more information about CORBA, see “Common Object Request Broker Architecture (CORBA)” on page 1153.

You can create a message model for your CORBA data in IBM App Connect Enterprise by using XML Schema; see:

- “Opening an existing XML schema file” on page 2242
- “Creating an XML schema file” on page 2243
- “Deleting an XML schema file” on page 2244
- “Generating HTML documentation from an XML Schema file” on page 2245
- “Working with XML Schema message model objects” on page 2245

Working with SAP data

You can create a message model for data from an SAP system by using DFDL or by using WebSphere Adapter Schema.

About this task

You can create a message model for your SAP data in IBM App Connect Enterprise by using DFDL; see:

- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230

Alternatively, you can create a message model for your SAP data in IBM App Connect Enterprise by using WebSphere Adapter Schema.

For more information about WebSphere Adapter for SAP Software, see “Overview of WebSphere Adapter for SAP Software” on page 1052.

Working with Siebel data

You can create a message model for data from a Siebel system by using WebSphere Adapter Schema.

About this task

You can create a message model for your Siebel data in IBM App Connect Enterprise by using WebSphere Adapter Schema.

For more information about WebSphere Adapter for Siebel Business Applications, see “Overview of WebSphere Adapter for Siebel Business Applications ” on page 1092.

Working with PeopleSoft data

You can create a message model for data from a PeopleSoft system by using WebSphere Adapter Schema.

About this task

You can create a message model for your PeopleSoft data in IBM App Connect Enterprise by using WebSphere Adapter Schema.

For more information about WebSphere Adapter for PeopleSoft Enterprise, see “Overview of WebSphere Adapter for PeopleSoft Enterprise” on page 1098.

Working with JD Edwards data

You can create a message model for data from a JD Edwards system by using WebSphere Adapter Schema.

About this task

You can create a message model for your JD Edwards data in IBM App Connect Enterprise by using WebSphere Adapter Schema.

For more information about WebSphere Adapter for JD Edwards EnterpriseOne, see “Overview of WebSphere Adapter for JD Edwards EnterpriseOne” on page 1102.

Working with database records

You can create a message model for data from database records by using XML Schema.

About this task

You can create a message model for your database record data in IBM App Connect Enterprise by using XML Schema; see:

- “Opening an existing XML schema file” on page 2242
- “Creating an XML schema file” on page 2243
- “Deleting an XML schema file” on page 2244
- “Generating HTML documentation from an XML Schema file” on page 2245
- “Working with XML Schema message model objects” on page 2245

Working with JSON data

Message modeling for JSON format data is supported by IBM App Connect Enterprise.

About this task

JSON (JavaScript Object Notation) is a simple data-interchange format based on a subset of the JavaScript programming language. A JSON message consists of name-value pairs (objects), and ordered collections of values (arrays). Objects, arrays, or both structures can be nested.

You can create a message model for your JSON data in IBM App Connect Enterprise by using JSON Schema.

For more information about JSON, see the JavaScript Object Notation (JSON) web site.

Working with MIME data

You can create a message model for MIME data by using XML Schema.

About this task

Multipurpose Internet Mail Extensions (MIME) is a message format that specifies message headers, but which leaves the message body as ASCII text.

For more information about MIME, see RFC 1521: MIME Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies.

You can create a message model for your MIME data in IBM App Connect Enterprise by using XML Schema; see:

- “Opening an existing XML schema file” on page 2242
- “Creating an XML schema file” on page 2243
- “Deleting an XML schema file” on page 2244
- “Generating HTML documentation from an XML Schema file” on page 2245
- “Working with XML Schema message model objects” on page 2245

How to model data with DFDL

Analyze data formats for which no model exists. Understand the structure, so that you can create the corresponding DFDL model.

Data modeling with DFDL has a strong analogy with programming. Suppose you want to learn a new programming language and write a program to solve a business problem. Take Java as an example. You buy a Java book and read up on the language theory. You get hold of a good Java editor and learn how to use it. But the hardest part is working out how to structure the program that solves your business problem. To get a head start, you might look at examples that other programmers created.

DFDL data modeling is the same. You can learn the theory about the modeling language and you can learn how to use an editor for that language. But the hardest part is looking at the actual data and working out how to go about creating the best model for it. If you are lucky the problem is solved in whole or in part by already having a model of the data in one format or another (metadata). For DFDL, IBM provides importers to convert certain data formats (for example, COBOL and C metadata) into DFDL schemas. But what do you do when you have no model to reference, just one or more examples of the data format? Formatted text messages, such as comma-separated values (CSV) messages, often have no model. You can learn how to analyze data formats in order to understand the structure and create the corresponding DFDL model.

The task consists of the following stages:

1. “Understanding the logical structure” on page 2214
2. “Configuring the DFDL annotations” on page 2215

3. “Organizing the DFDL model” on page 2219

Understanding the logical structure

The first stage of an approach to modeling data by using DFDL involves examining the logical structure of your data.

Procedure

1. Identify complex structures.

   Complex structures correspond to the complex types in the model. There will be an overall complex type for the entire data itself. If the data contains substructures, each substructure has a complex type. For example, each structure level in a COBOL copy book, or each different row in a CSV message, corresponds to an element of complex type.
2. Identify simple items.

   Simple items occur within each complex type, and each has a logical data type. Simple items correspond to simple elements. For example, each field in a COBOL copy book with a PIC clause, or each comma-separated value in a CSV message, corresponds to an element of simple type.
3. Identify structure ordering.

   Ordering determines whether the group within a complex type is a sequence or a choice. In a choice, only one of the listed items can occur. Examples are C unions and COBOL REDEFINES.
4. Identify structure and item cardinality.

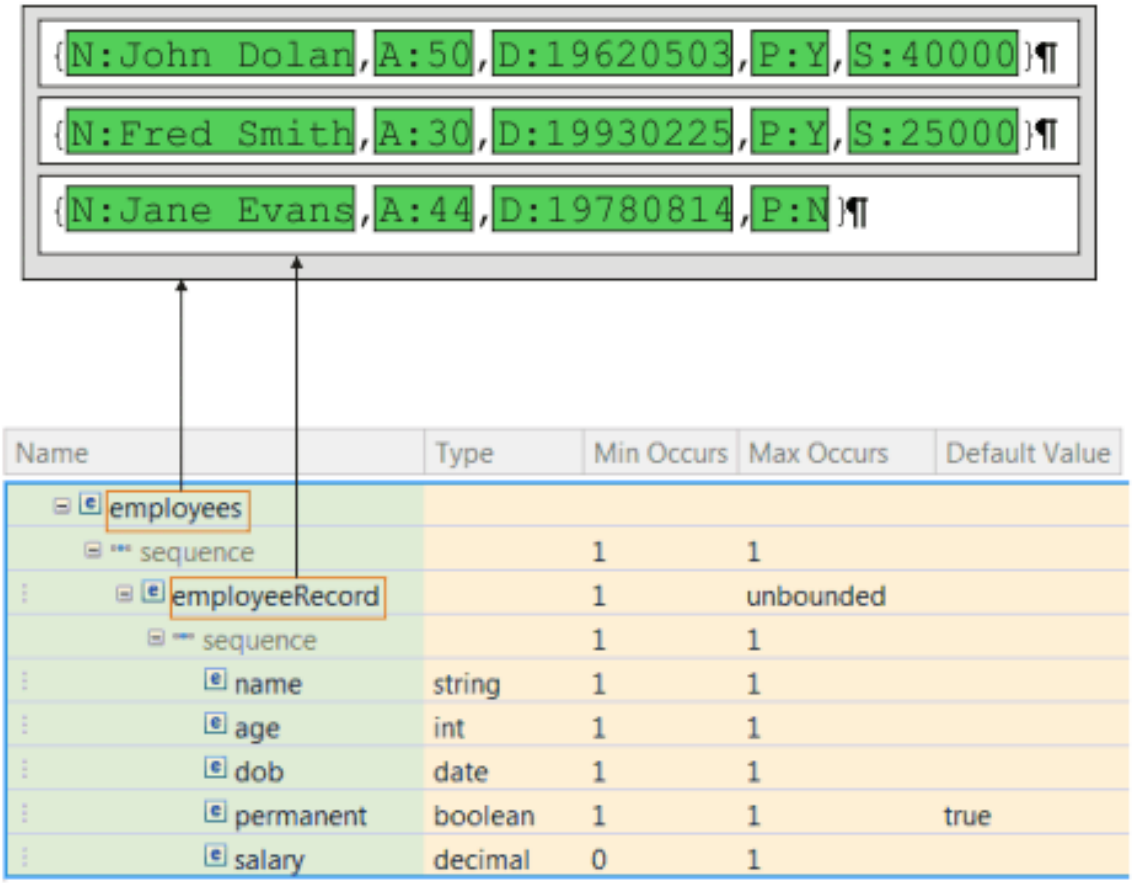
   Cardinality provides the values for the `minOccurs` and `maxOccurs` logical properties of your elements.

   - Is an element required (`minOccurs != 0`) or optional (`minOccurs = 0`)?
   - Is an element an array (`maxOccurs > 1`)?
   - If so are there a fixed number of occurrences (`minOccurs = maxOccurs`) or a variable number of occurrences (`minOccurs != maxOccurs`)?
   - Can the number of occurrences be unlimited (`maxOccurs = unbounded`)?
5. Identify nillable items and default values.

   Some elements might need to carry a special out-of-range value, in which case they must be nillable. For example, a numeric field in a COBOL copy book might sometimes be set to SPACES, which is not legal for a DFDL number. Some required elements might be empty in the data, in which case a default value can be provided.
6. Consider whether any components can be reused.

   If any of the types are common, consider creating global complex or simple types. If any of the elements are common, consider creating global elements.

Example



As an example, the picture shows a file of employee records. This file could be modeled using DFDL as an overall complex element `employees` that contains a complex element `employeeRecord`. The `employeeRecord` element repeats an arbitrary number of times, hence maxOccurs is set to unbounded.

The `employeeRecord` is a sequence of simple elements:

- `name` of type `xs:string`
- `age` of type `xs:int`
- `dob` of type `xs:date`
- `permanent` of type `xs:boolean`
- `salary` of type `xs:decimal`.

The `salary` element is present only when `permanent` is Y, so it is optional and has minOccurs `0`. All the other simple elements are required and have minOccurs `1`.

What to do next

The next stage is to configure the DFDL annotations: “Configuring the DFDL annotations” on page 2215

Configuring the DFDL annotations

The second stage of an approach to modeling data using DFDL involves adding DFDL annotations to the logical structure that you established. The DFDL annotations describe the physical format of the components.

Before you begin

Follow the guidance in “Understanding the logical structure” on page 2214.

About this task

Determine the characteristics of your components.

Procedure

1. All elements (simple and complex)
   a) Does the element have any delimiters, that is, an initiator or a terminator? If so what is the encoding, and are they present when the element is empty or nil?

      This characteristic determines the dfdl:initiator, dfdl:terminator, dfdl:encoding and associated properties.

   b) How is the content of the element established?

      This characteristic determines the dfdl:lengthKind and associated properties:

      - `explicit` for a fixed length.
      - `prefixed` if there is a length prefix.
      - `delimited` if bounded by a delimiter.
      - `pattern` to use a regular expression.
      - `implicit` if the length is determined by its type.
      - `endOfParent`, if bounded by its parent.

   c) If the element is optional or is an array, how is the number of occurrences established?

      This characteristic determines the dfdl:occursCountKind and associated properties.

   d) Are there any alignment rules to apply?

      This characteristic usually occurs only for binary data, and determines the dfdl:alignment, dfdl:fillByte, and associated properties.

   e) How is any nil value described?

      This characteristic determines the dfdl:nilKind, dfdl:nilValue and associated properties.

   f) Is an assert or discriminator needed to establish whether the element exists?
2. Simple elements
   a) Is the element text or binary representation? The representation and simple type determines which other properties must be set.

      - For text, the properties are dfdl:encoding, and the several DFDL text-related properties.
      - For binary, the properties are dfdl:byteOrder, and the several DFDL binary-related properties.

   b) For text formats, is an escape scheme needed?

      This characteristic determines whether a dfdl:defineEscapeScheme annotation is needed, and if so a dfdl:escapeSchemaRef to reference it.

   c) If global simple types are identified, decide whether the simple type can carry some of the properties rather than the element, thus creating reusable physical types.

3. Sequences
   a) Is the sequence ordered or unordered?

      This characteristic determines the dfdl:sequenceKind property.

   b) Does it have a separator that is used to delimit its child elements, and if so is the separator's position `infix`, `prefix` or `postfix`? Are separators sometimes suppressed (for example, when optional elements are missing)?

      These characteristics determine the dfdl:encoding, dfdl:separator, dfdl:separatorPosition, and dfdl:separatorSuppressionPolicy properties.

   c) Do all the child elements of the sequence have unique initiators that can identify that they exist?

      This characteristic determines the dfdl:initiatedContent property.

   d) Does the sequence itself have an initiator or a terminator?

      This characteristic determines the dfdl:initiator, dfdl:terminator, dfdl:encoding, and associated properties.

4. Choices
   a) Is the choice one where all the branches must occupy the same length or not?

      This characteristic determines the dfdl:choiceLengthKind and associated properties.

   b) Do all the branches of the choice have unique initiators that can identify which one appears?

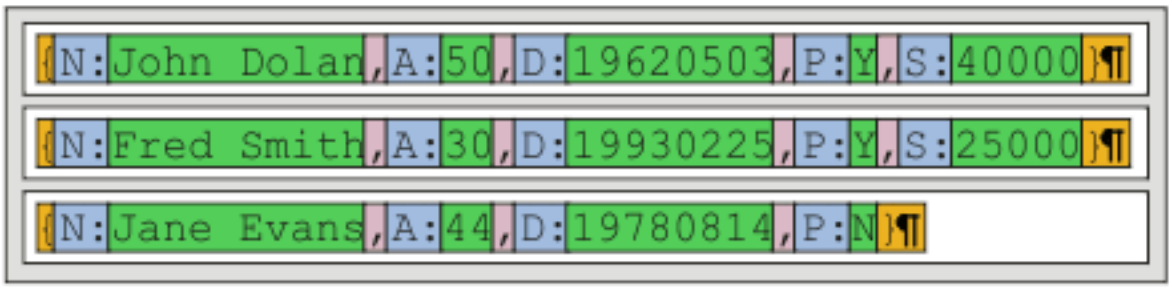
      This characteristic determines the dfdl:initiatedContent property.

   c) Are discriminators needed on the branches to establish which one appears?

   d) Does the choice itself have an initiator or a terminator?

      This characteristic determines the dfdl:initiator, dfdl:terminator, dfdl:encoding, and associated properties.

Example



Name	Type	Min Occurs	Max Occurs
employees			
sequence		1	1
employeeRecord		1	unbounded
sequence		1	1
name	string	1	1
age	int	1	1
dob	date	1	1
permanent	boolean	1	1
salary	decimal	0	1

Property	Value
Comment	
General	
Encoding (code p	US-ASCII
Byte Order	bigEndian
Content	
Length Kind	implicit
Occurrences	
Min Occurs	1
Max Occurs	unbounded
Occurs Count Kind	implicit
Delimiters	
Initiator	{{
Terminator	}%CR;%LF;

To continue with the example of a file of employee records, in which all data is text, with a dfdl:encoding of ASCII.

- The `employees` element does not have an initiator or a terminator so dfdl:initiator and dfdl:terminator are set to empty string ''. Its length is determined by its child elements, so dfdl:lengthKind is `implicit`.
- The sequence for `employees` has dfdl:sequenceKind `ordered`, because its child components always appear in the order specified.
- The `employeeRecords` element starts with `{`, so has a dfdl:initiator `{{`. (Two opening braces (`{{`) are needed to prevent the DFDL initiator from being misinterpreted as a DFDL expression.) The element ends with `}` and CR/LF, and so has dfdl:terminator `}%CR;%LF;`. (An alternative is to model the `}` and CR/LF as the separator of the parent sequence.) Again, its length is determined by its child elements, so dfdl:lengthKind is `implicit`.
- Its sequence has dfdl:sequenceKind `ordered`.
- Each simple element has dfdl:representation `text`. Each has a unique start tag that is used as the dfdl:initiator. Each has a variable length value that is delimited either by ',' or, if the element is last in the record, by `}` and CR/LF. Consequently, dfdl:lengthKind is `delimited`.
- The `','` delimiter is best modeled as the dfdl:separator of the parent sequence, and not as the dfdl:terminator of the element. The dfdl:separatorPosition is `infix` meaning that the `','` occurs only between the child elements in the sequence.
- Because each simple element has an initiator, dfdl:initiatedContent `yes` must be set on the parent sequence. This means the parser uses the initiator to positively identify each element.
- Although the `salary` element is optional, no DFDL discriminator is needed because the DFDL parser deduces that it is missing when it finds the `employeeRecord` terminator. When `salary` is missing,

notice that the `','` before it is suppressed. That is modeled using dfdl:separatorSuppressionPolicy `trailingEmpty` on the parent sequence.

- Several other text-related properties must be set for each simple element. These properties control whether any padding or trimming takes place, and whether an escape scheme is in use to prevent data characters being interpreted as delimiters.
- Several other type-specific properties must be set for each simple element, to control the interpretation of the data value. For example, the `permanent` element is type xs:boolean and so needs dfdl:textBooleanTrueRep Y and dfdl:textBooleanFalseRep N.

What to do next

The next stage is to organize the DFDL model: "Organizing the DFDL model" on page 2219.

Organizing the DFDL model

The third stage of an approach to modeling data by using DFDL involves organizing the DFDL model so that common DFDL property values are declared in a single place and act as defaults for all the components in the schema.

Before you begin

Before you start:

Follow the guidance in "Understanding the logical structure" on page 2214 and "Configuring the DFDL annotations" on page 2215.

About this task

All DFDL schemas that are created by the IBM DFDL wizards and importers follow the practices as described here.

Procedure

- Set up common DFDL property defaults by using a dfdl:format annotation at the top level of the schema.

  A DFDL schema can be set up so that DFDL properties that are common to multiple schema components need be declared only once, by using a dfdl:format annotation at the top level of the schema itself to declare properties. These properties effectively act as defaults for all components in the schema. (DFDL has no built-in property defaults.)
- A further refinement is to place those properties in a separate DFDL schema for reuse in other related DFDL schemas.

  Create a separate DFDL schema to contain these common DFDL properties, and place them inside a dfdl:defineFormat annotation. This schema is then included into the main DFDL schema through an XSD include or import. The dfdl:format annotation at the top level of the main schema instead uses dfdl:ref to refer to the dfdl:defineFormat. Much like a macro expansion, properties are pulled from the dfdl:defineFormat onto the dfdl:format, where they then act as defaults for all components in

the schema in the way described above. In this way, common DFDL properties can be shared across multiple related DFDL schemas.

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/" >
      <dfdl:defineFormat name="myDefaults" >
         <!- Declare common DFDL property values -->
         <dfdl:format encoding="ASCII" representation="text" ... />
      </dfdl:defineFormat>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

Figure 36. defaults.xsd

- Now, you need only to set a handful of properties directly on each object to complete configuration.

1.
```
<xs:schema>
  <xs:include schemaLocation="defaults.xsd" />

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/" >
      <!- Apply common DFDL property values as defaults -->
      <dfdl:format ref="myDefaults" />
    </xs:appinfo>
  </xs:annotation>

  <!- Add only DFDL properties that differ from the defaults -->
  <xs:element name="employeeRecord" maxOccurs="unbounded" dfdl:lengthKind="implicit"
                                    dfdl:initiator="{{" dfdl:terminator="}%CR;%LF;" >
    <xs:complexType>
         ...
    </xs:complexType>
  </xs:element>

  ...

</xs:schema>
```

Figure 37. employees.xsd

Working with DFDL schema files

Create, open, modify or delete a DFDL schema file.

About this task

This topic area describes the tasks that are involved in working with a DFDL schema file:

- “Opening an existing DFDL schema file” on page 2221
- “Creating a DFDL schema file” on page 2223
- “Importing a DFDL schema file” on page 2221
- “Editing a DFDL schema file” on page 2225
- “Testing a DFDL schema file” on page 2226
- “Deleting a DFDL schema file” on page 2229
- “Working with DFDL message model objects” on page 2230
- “Developing independent DFDL applications” on page 2239

Opening an existing DFDL schema file

Open an existing DFDL schema file in the DFDL schema editor to view or edit the contents of the file.

About this task

To open an existing DFDL schema file:

Procedure

1. Right-click the DFDL schema file (file extension `*.xsd`) that you want to open, and select **Open**.
   The DFDL schema file opens in the DFDL schema editor.

   Tip: The Eclipse framework lets you open resource files with other editors. However, you are advised to use the DFDL schema editor to work with DFDL schema files. This editor correctly validates any changes that are made to your DFDL schema files. Other editors might not correctly validate files.
2. View or edit the data in your DFDL schema file.
   For more information about using the DFDL schema editor, see DFDL schema editor.

Importing a DFDL schema file

You can import DFDL schema files that were created outside of the IBM App Connect Enterprise Toolkit, and use them in developing message flow applications.

About this task

Note: For information about how to create a DFDL schema file from an existing resource, see "Creating a DFDL schema file by using the New Message Model wizard" on page 2223.

Complete the following tasks to import your DFDL schema files:

- "Importing DFDL schema files into the workspace" on page 2221
- "Adding IBM App Connect Enterprise annotations to imported DFDL schema files" on page 2221

Importing DFDL schema files into the workspace

Before you begin

- You must have created a DFDL schema file that conforms to the DFDL specification.
- You must have the IBM App Connect Enterprise Toolkit open in the Integration Development perspective.

About this task

You can use the drag-and-drop method to import DFDL schema files from your file system into the IBM App Connect Enterprise Toolkit. In the Application Development view, drag the DFDL schema files that you are importing to the application or library. Do not drag them onto a blank area in the Application Development view.

Adding IBM App Connect Enterprise annotations to imported DFDL schema files

If you have imported DFDL schema files that were created outside of the IBM App Connect Enterprise Toolkit, you must add the required DFDL annotations for use with IBM App Connect Enterprise.

Before you begin

Before you start:

Complete the following task:

- "Importing DFDL schema files into the workspace" on page 2221

About this task

If you want to use a DFDL schema file that has been created outside of the IBM App Connect Enterprise Toolkit, you must add the required IBM App Connect Enterprise annotations before you can use the DFDL schema file in the development of your message flow applications. You may encounter errors if you use a DFDL schema file that does not contain the required IBM App Connect Enterprise annotations.

To add the required IBM App Connect Enterprise annotations to your DFDL schema file:

Procedure

1. In the Application Development view, right-click the imported DFDL schema file (file extension `*.xsd`) that you want to open, and select **Open with** > **XML Editor**.

   The DFDL schema file opens in the XML editor.
2. Add the namespace `xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions"` by completing the following steps:
   a) In the XML editor, right-click anywhere within your DFDL schema file and select **Edit Namespaces...**.

      The **Edit Schema Information** window opens.
   b) In the **Edit Schema Information** window, click **Add...**.

      The **Add Namespace Declarations** window opens.
   c) In the **Add Namespace Declarations** window, select **Specify New Namespace**.
   d) In the Prefix field, enter `ibmSchExtn`. In the Namespace Name field, enter `http://www.ibm.com/schema/extensions`, then click **OK**.
   e) In the **Edit Schema Information** window, click **OK**.

   The namespace is added to your DFDL schema file, and the **Edit Schema Information** window closes.
3. Add the annotation, `ibmSchExtn:docRoot="true"` to the top-level element of your message model by completing the following steps:
   a) In the XML editor, locate the top-level element of your message model. Right-click this element and select **Add Attribute** > **Default**.

      An attribute containing default values is added to your message model.
   b) Locate the default attribute that you created in the last step. Right-click this attribute, and select **Edit Attribute...**.

      The **Edit Attribute** window opens.
   c) In the Name field, enter `ibmSchExtn:docRoot`. In the Value field, enter `true`. Click **OK**.

   The annotation is added to your DFDL schema file, and the **Edit Attribute** window closes.
4. In the Navigator view, right-click your DFDL schema file (file extension `*.xsd`), and select **Open with** > **DFDL Editor**.

   The DFDL schema file opens in the DFDL schema editor.

What to do next

Next:

- You can continue to edit your DFDL schema by using the DFDL schema editor.
- You can test your message model, see "Testing a DFDL schema file" on page 2226.

Reimporting IBM supplied message models

If you have modified one or more IBM supplied message models, you can reimport them to overwrite your changes, restoring them to their default state.

About this task

Procedure

1. Click **File** > **New** > **Other**.
   A window opens in which you can select a wizard.
2. Expand **App Connect Enterprise - Application Development**, select **Message Model ...**, click **Next**.
   The **New Message Model** wizard opens.
3. Select **IBM supplied**, then click **Next**.
4. Select **Replace the imported IBM supplied definitions**. Click **Next**,
5. Step through the remainder of the wizard, clicking **Next** to move to a new panel, and then click **Finish** when you have selected the IBM supplied message model that you want to replace.

Results

The IBM supplied message model that you selected is reimported. It is displayed in the **XML and DFDL Schemas** folder of your project and is opened in the DFDL schema editor.

What to do next

Next:

- You can continue to edit your DFDL schema by using the DFDL schema editor.
- You can test your message model, as described in "Testing a DFDL schema file" on page 2226.

Creating a DFDL schema file

You can create DFDL schema files to model your text and binary formatted data.

About this task

You can use DFDL schema to model various data formats, including:

- Comma Separated Value (CSV)
- Record-oriented text
- Custom text or binary
- COBOL data structures
- C data structures

Creating a DFDL schema file by using the New Message Model wizard

Use the **New Message Model** wizard to create DFDL schema files.

About this task

You can use DFDL schema to model various data formats, including:

- Comma Separated Value (CSV).
- Record-oriented.
- COBOL data structures. For an overview of how to use the COBOL importer, go to the Rational Application Developer for WebSphere Version 8.5.5 product documentation online and search for *COBOL Importer overview*.

- C Header files. Some restrictions apply; see Importing from C (DFDL): restrictions.
- Custom text or binary.

Procedure

1. Open the New Message Model wizard by using one of the following methods:
   - In the navigator, click **New**, then click **Message Model ...**.
   - Right-click the white space of the navigator, then click **New** > **Message Model ...**.
2. Step through the wizard, clicking **Next** to move to a new panel, and clicking **Finish** when you have entered the information required to create your DFDL schema.

   Your new DFDL schema is created, and is displayed in your application or library.

What to do next

Next:

The schema file is opened in the DFDL editor, and you can edit it immediately.

Creating a DFDL schema file from a C header file
Create a DFDL schema file from a C header file.

About this task

Using the wizard, you can create a DFDL schema file by:

- Importing a single C header file. Some restrictions apply; see Importing from C (DFDL): restrictions.
- Importing or replacing the IBM-supplied DFDL C format schema.

Tip: If you want to import many header files at once from the command line, see Message sets: Importing from C header files to create message definitions. This procedure produces message definitions for use in the MRM domain, not the DFDL domain.

Procedure

1. Click **File** > **New** > **Other** > **Message Model**, and select **C header file**.
2. Step through the wizard to create the DFDL schema file.

Creating a DFDL schema file from a COBOL resource
Create a DFDL schema file from an existing COBOL resource.

Before you begin

Linux The COBOL importer requires 32-bit versions of the Linux operating system libraries. Some of these libraries are not installed by default on the following distributions:

Red Hat Enterprise Linux V6 64-bit
Ubuntu 10.4 or later, 32-bit and 64-bit

For details of which libraries to install, see https://www.ibm.com/support/docview.wss?uid=swg21512074

About this task

Using the wizard, you can create a DFDL schema file by:

- Importing a COBOL copybook or program.

  Note: The copybook must not contain field names that are COBOL reserved keywords. Also, the COBOL importer can only import files that have file names of 60 characters long or less. If you attempt to

import a COBOL file with a name longer than 60 characters, the COBOL importer fails with the message `IWAA0652E: File name cannot be longer than 60 characters`.

- Importing or replacing the IBM-supplied DFDL COBOL format schema.

Procedure

1. Click **File** > **New** > **Other** > **Message Model**, and select **COBOL**.
2. Step through the wizard to create the DFDL schema file.

Editing a DFDL schema file

Use the DFDL schema file editor to change an existing file.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. Click in the Outline view to select the category that you want to work with.
   For example, to edit message elements, select a child of the **Messages** item.

   The selected item is displayed in a table in the main editing area. Click **Show all sections** on the main toolbar of the editor to display all sections, including empty ones.
2. Use the buttons immediately above the table to modify items in the selected category, for example, deleting or adding elements.

   Different buttons are available, depending on the category that you are working with.
3. Use the **Representation Properties** tab on the right of the editor to modify properties.

   Tip:

   - You can select multiple objects in the Editor, but you see properties for only the first item that you selected.
   - Use **Filter** to find the property that you want to change.
   - Click **Show Only Basic Properties** to filter properties. If only basic properties are displayed, click **Show All Properties** to display all properties
   - To unset a property value, use the **Clear local property value** and **Clear all local properties** buttons to the right of the filter bar and filter actions. Using the **Delete** key can result in the property being set to an empty string, rather than being unset.
   - The value that is entered in the Sample Value column in the main editor view, or the Sample value text field in the properties, is the lexical representation of the corresponding XML Schema type. This value is what will appear in the logical instance that is generated. Information on the lexical representation of the different XML Schema types can be obtained from XML Schema Part 2: Datatypes Second Edition.
4. Save your file.

   Errors or warnings are displayed in the Problems view.

What to do next

Next:

Test your DFDL schema file.

Testing a DFDL schema file

When you have defined your DFDL schemas, test them in the editor before they are used on a production system.

Before you begin

Create or open a DFDL schema file.

About this task

You can use the DFDL Test Perspective to test-parse and test-serialize sample data against your DFDL schemas.

You can test a DFDL schema in one or all of the following ways:

- Generate a logical instance document from your DFDL schema; see "Testing a DFDL schema by creating a logical instance" on page 2226.
- Test-parse sample data against your DFDL schema; see "Testing a DFDL schema by parsing test input data" on page 2227.
- You can test-serialize from a logical instance using data from your DFDL schema; see "Testing a DFDL schema by serializing test output data" on page 2228.

What to do next

Next:

- You can continue to edit your DFDL schema by using the DFDL schema editor.

Testing a DFDL schema by creating a logical instance

You can create a logical instance of messages that are modeled by your DFDL schemas by using the DFDL Test Perspective.

Before you begin

Create or open a DFDL schema file.

About this task

You can use the DFDL Test Perspective to generate a logical instance from your selected DFDL schema. The data used to create the logical instance comes from the "Sample Test Data" values in the schema.

Procedure

1. Open the DFDL schema that you want to test.
2. Right-click the document root that you want to test, and click **Create a logical instance for the DFDL schema**.

   A dialog is displayed that asks you whether you want to open the DFDL Test Perspective.
3. Click **Yes**.

   The DFDL Test Perspective opens, and your generated sample message is displayed in the **DFDL Test - Logical Instance** view.
4. View your sample message as a logical tree by clicking the **Tree View** tab, or view your sample message as XML by clicking the **XML view** tab.

What to do next

Next:

- You can test-serialize the logical instance data that you have created; see "Testing a DFDL schema by serializing test output data" on page 2228.

- You can continue to edit your DFDL schema by using the DFDL schema editor.
- You can test-parse sample data against your DFDL schemas; see "Testing a DFDL schema by parsing test input data" on page 2227.
- If the DFDL schema is complete, you can add it to a BAR file for deployment; see Creating a BAR file.

Testing a DFDL schema by parsing test input data

You can test-parse sample data against your DFDL message model by using the DFDL Test Perspective.

Before you begin

Create or open a DFDL schema file.

About this task

You can use the DFDL Test Perspective to test-parse sample data against your selected DFDL message, to verify that the DFDL schema is correct.

Procedure

1. Open the DFDL schema that you want to test.
2. In the DFDL schema editor, expand the **Messages** section.
3. Select the message that you want to test.
4. Click **Test parse model**, or right-click the message that you want to test and click **Test parse model**.
   The **Test Parse Model** window opens.
5. In the **Parser Input** section, select the location of the input data that you want to test-parse:
   - If you have not previously serialized a test message:
     a. Select **Content from the file...**
     b. In the **Input file name** field, enter the location of your input data file, or click **Browse** to locate it.
   - If you have previously serialized a test message, select from the following options:
     – Select **Content from the file...** to test-parse the contents of an input file, as described in the previous step.
     – Select **Content from 'DFDL Test - Serialize' view** to test-parse the data that is displayed in the **DFDL Test - Serialize** view.
6. Enter **Runtime encoding options** as required.
7. Optional: Select **Validate data against schema** to check for validation errors.
8. Click **OK**, and confirm that you want to open the DFDL Test Perspective.

Results

- The DFDL Test Perspective opens, and the results of your test-parse are displayed in the **DFDL Test - Parse** view.
- You can view a log of parser actions in the **DFDL Test - Trace** view.
- You can view the logical instance in the **DFDL Test - Logical Instance** view, as a tree, or as XML.

What to do next

Next:

- You can continue to edit your DFDL schema by using the DFDL schema editor.
- You can generate a structural preview of a message from your DFDL schema; see "Testing a DFDL schema by creating a logical instance" on page 2226.
- You can test-serialize data from your DFDL schema, see "Testing a DFDL schema by serializing test output data" on page 2228.

- You can change the following settings, and then test-parse again by clicking the **Run Parser** button in the **DFDL Test - Parse** view.
  - In the **DFDL Test - Parse** view, you can change the input data and encoding.
  - In the Editor, you can select a different message to be test-parsed.

  All other settings from the last test-parse are retained.
- If the DFDL schema is complete, you can add it to a BAR file for deployment; see Creating a BAR file.

Debugging

About this task

If an error occurs during parsing, the following resources are available to help correct the model:

- An error message is displayed. The message summarizes the problem, and contains links to view the trace and partial logical instance that was created.
- The **DFDL Test - Parse** view shows the location of the error in the data. At the top of this view is a hyperlink that opens the **DFDL Test - Trace** view.
- The object in error is marked in the Editor with the following marker:  . The error marker has hover help that provides information about the error. To clear the error marker, correct the model and then parse the data again. The error marker is also removed when you close the DFDL editor.
- The **DFDL Test - Logical Instance** view shows the infoset parsed up to the error.

Testing a DFDL schema by serializing test output data

You can produce serialized test data from your DFDL schemas by using the DFDL Test Perspective.

Before you begin

Create or open a DFDL schema file.

About this task

You can use the DFDL Test Perspective to test-serialize sample data against your selected DFDL message to verify that the DFDL schema is correct.

Procedure

1. Open the DFDL schema that you want to test.
2. In the DFDL schema editor, expand the **Messages** section.
3. Select the message that you want to test.
4. Click **Test serialize model**, or right-click the message that you want to test and click **Test serialize model**.

   The **Test Serialize Model** window opens.
5. In the **Serializer Input** section, select the location of the logical instance data that you want to test-serialize:
   - If you have a populated 'DFDL Test - Logical Instance' view, select **Content from 'DFDL Test - Logical Instance'** to test-serialize the data that is displayed there.
   - If you do not have a populated 'DFDL Test - Logical Instance' view:
     a. Select **Content from a logical instance file**.
     b. In the **Input file name** field, enter the location of your file of logical instance data, or click **Browse** to locate it.
6. Enter **Runtime encoding options** as required.
7. Optional: Select **Validate data against schema** to check for validation errors.
8. If prompted, confirm that you want to open the DFDL Test Perspective.

Results

- The DFDL Test Perspective opens, and the results of your test-serialize are displayed in the **DFDL Test - Serialize** view.
- You can view a log of parser actions in the **DFDL Test - Trace** view.
- You can view the logical instance in the **DFDL Test - Logical Instance** view, as a tree, or as XML.
- You can change the following settings, and then test-serialize again by clicking the **Run Serializer** button in the **DFDL Test - Serialize** view.
  - In the **DFDL Test - Serialize** view, you can change the logical instance and encoding.

  All other settings from the last test-serialize are retained.

What to do next

Next:

- You can continue to edit your DFDL schema by using the DFDL schema editor.
- You can generate a structural preview of a message from your DFDL schema; see "Testing a DFDL schema by creating a logical instance" on page 2226.
- You can test-parse sample data against your DFDL schemas, see "Testing a DFDL schema by parsing test input data" on page 2227.
- You can change the following settings, and then test-serialize again by clicking the **Run Serializer** button in the **DFDL Test - Serialize** view.
  - In the **DFDL Test - Serialize** view, you can change the logical instance and encoding.

  All other settings from the last test-serialize are retained.
- If the DFDL schema is complete, you can add it to a BAR file for deployment; see Creating a BAR file.

Debugging

About this task

If an error occurs during parsing, the following resources are available to help correct the model:

- An error message is displayed. The message summarizes the problem, and contains links to view the trace output.
- The **DFDL Test - Serialize** view shows the location of the error in the data. At the top of this view is a hyperlink that opens the **DFDL Test - Trace** view.

Deleting a DFDL schema file

You can delete a DFDL schema file.

Before you begin

Create a DFDL schema file.

Procedure

1. Right-click the DFDL schema file (file extension `.xsd`) that you want to delete, then click **Delete**.

   Alternatively, select the DFDL schema file that you want to delete, then, from the menu bar, click **Edit** > **Delete**, or press the Delete key.
2. In the **Confirm Resource Delete** window, click **Yes** to delete the DFDL schema file.

   Important: this action cannot be undone.

Results

Important: All files and objects that are associated with the DFDL schema file are deleted.

Working with DFDL message model objects

Add, configure, and delete objects.

About this task

The following tasks are described:

- "Adding DFDL message model objects" on page 2230
- "Configuring DFDL message model objects" on page 2235
- "Deleting objects" on page 2238
- "Linking from one DFDL schema file to another" on page 2238

Adding DFDL message model objects

Use the DFDL schema editor to add DFDL message model objects.

About this task

The following tasks are described:

- "Adding DFDL format annotations" on page 2230
- "Adding DFDL escape schemes" on page 2231
- "Adding a message" on page 2231
- "Adding a global element" on page 2232
- "Adding a local element" on page 2232
- "Adding an element reference" on page 2232
- "Adding a simple type" on page 2233
- "Adding a global complex type" on page 2233
- "Adding a global group" on page 2233
- "Adding a local sequence" on page 2234
- "Adding a group reference" on page 2234
- "Adding groups" on page 2233
- "Adding a user-defined variable" on page 2234

Adding DFDL format annotations

Data formats are collections of DFDL properties that can be referenced by objects in the DFDL schema such as elements and sequences.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **DFDL** and then click **Add a Definition Format** to add an existing format. If a format does not already exist, click **Add a default format**.
2. Configure the scheme in the **Representation Properties** .

What to do next

Next:

To apply the format to an object in the schema, select the format in the **Data Format Reference** property.

Adding DFDL escape schemes
An escape scheme tells DFDL to treat characters literally, rather than as a separator or terminator. For example, you might need to escape commas in a CSV file.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

About this task
Add an escape scheme, and then refer to it using the "Escape Scheme Reference" property on the relevant elements.

Procedure
1. In the Outline view, expand **DFDL** > **Escape Schemes**.
2. In the DFDL schema editor, click the icon to add an escape scheme to the schema, and type a name when prompted.
3. Configure the scheme in the **Representation Properties** :
   a) `Escape Kind` :
      Possible values:
      - `escapeCharacter`: A single escape character that causes only the next character to be interpreted literally. The escape character itself is escaped by the `Escape Escape` character.
      - `escapeBlock`: A pair of escape strings that causes the enclosed group of characters to be interpreted literally. The ending escape string is escaped by the `Escape Escape` character.
   b) **Escape Character**: if the escape kind is `escapeCharacter`, type a single character that escapes the following character.
   c) **Escape Block Start**: type the string of characters that mark the beginning of the escape block.
   d) **Escape Block End**: type the string of characters that mark the end of the escape block.
   e) **Escape Escape**: specify the character that escapes the escape character or string.
   f) **Extra Escaped Character**: optionally, specify any other characters that must be escaped when serializing.
   g) **Generate Escape Block**: if the escape kind is `escapeBlock`, define when escaping is used during serialization.

Adding a message
Add a global element that represents an entire message.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure
1. In the Outline view, right-click **Messages**, and then select **Add**.
2. Type a name for the message.
3. Specify whether the content of the message is a sequence or a choice.

Adding elements
Define global and local elements.

About this task

By using an *element reference*, a *global* element can be reused across messages, types, groups, and other elements.

A *local* element is available only where it is declared and cannot be reused.

Adding a global element
Use the outline view of the DFDL schema editor to create a global element.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **Elements**, and then select **Add a Global Element**.
2. Type a name for the element.

What to do next

Next:

Configure the global element; see "Editing a DFDL schema file" on page 2225.

Adding a local element
Use the outline view of the DFDL schema editor to add a local element.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. If necessary, in the Outline view, select the appropriate part of the schema.

   This action displays the selected area in the main editing area.
2. In the main editing area, right-click the parent element and select either **Add a Local Element** or **Add Complex local element**. If you are adding a complex local element, specify whether it is a `sequence` or a `choice` element.

   The new element is created as the last child of the parent element.
3. Optional: Use the **Up** and **Down** arrows above the table that you are working in to reposition the new element.

Adding an element reference
Reuse a global element that you have defined.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

About this task

After you have defined a global element, you can include a reference to it in a sequence or choice. You define the element only once, and can then easily reuse it. Any changes that you make to the element are automatically reflected in other objects that contain a reference to the element.

Procedure

1. In the Outline, select the message, element, type, or group that you want to add an element reference to.
2. Right-click the selected object in the Editor area of the screen and select **Add Element Reference**.
3. Select the appropriate element from the list.

   A reference to the element is inserted. An arrow on the icon next to the element name denotes that it is a reference to another element.

Adding a simple type
Create a global simple type.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **Simple Types**, and then select **Add**.
2. Type a name and select an existing that the new type inherits its properties from.
3. Edit the properties of the new type as appropriate.

Adding a global complex type
Create a complex type.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **Complex Types**, and then select **Add**.
2. Type a name for the new type.
3. In the Editor area, add elements to the new type as appropriate.

Adding groups
Define a group of elements that can be used in complex type definitions.

About this task
A group can be either of the following compositions:

Sequence
All objects in the group can appear in the data.

Choice
Only one of the objects in the group can ever appear in the data.

Both compositions of group can be either global or local.

A *global* group has a name. Using a group reference, it can be reused across messages, elements, types, and other groups.

A *local* group has no name. In the editor, it is shown as either *sequence* or *choice*. It is available only where it is declared and cannot be reused.

Adding a global group
Create a named group of elements that you can reuse in complex definitions.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **Groups** > **Add a Group**, and then type the group name.
   The group is displayed in the editor area of the window.
2. Add and configure elements as appropriate.

Adding a local sequence
Add an unnamed sequence.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, select the object that you want to add a group to.
2. In the editor area, right-click the element that you want to add a group to, and select **Add Sequence**.
   The new group is placed as the last item. You can reorder it, and add elements to it.

Adding a local choice
Add an unnamed choice.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, select the object that you want to add a group to.
2. In the editor area, right-click the element that you want to add a group to, and select **Add Choice**.
   The new group is placed as the last item. You can reorder it, and add elements to it.

Adding a group reference
Reuse a global group that you have defined.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

About this task
After you have defined a group, you can include a reference to it in other objects. You define the group only once, and can then easily reuse it. Any changes that you make to the group are automatically reflected in other objects that contain a reference to the group.

Procedure

1. In the Outline, select the message, element, type, or group that you want to add a group reference to.
2. Right-click in the Editor area of the screen and select **Add Group Reference**.
3. Select the appropriate group from the list.
   A reference to the group is inserted. An arrow by the group name denotes that it is a reference to another group.

Adding a user-defined variable
You can use a user-defined variable to specify a value in your DFDL schema that is used in parsing your message, but varies between messages. Variables are defined as top-level annotations in a schema. A

variable holds a value that can be set once, and then accessed from DFDL expressions in property values, asserts, and discriminators.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, right-click **Variables**, then select **Add a Variable Definition**.
2. Type a name and select a data type for your user-defined variable.
3. Edit the default value of your new variable.
4. Save your message model.

Results

Your user-defined variable is added to the DFDL schema.

What to do next

Add your user-defined variable to a DFDL schema object. See "Setting variables on a schema object" on page 2237.

Configuring DFDL message model objects

Use the DFDL schema editor to configure DFDL message model objects.

About this task

The following tasks are described:

Renaming objects

Rename an existing object, such as a message or complex or simple type.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. Locate the object in the editor area.
2. Click the name of the object, and type a new name.

Reordering objects

Move a local element, sequence, or choice up or down the schema.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. Locate the object in the editor area.
2. Right-click the object, and select **Move Up** or **Move Down**.
   You can also use the icons that are below the main toolbar, in the editor area of the screen.

Changing the type of an element
Change the type of an element, for example from `integer` to `float`.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, select the parent of the element that you want to change.
   For example, to change a local element in a message, select the parent message.
2. In the editor area, click the **Type** property of the object, and select the appropriate type from the list.

   Tip:

   - Click **Anonymous** to define a new local complex type.
   - Click **Browse** to open a **Type Selection** window. From here, if the type that you require is not found, you can add a reference to another schema.

Configuring object properties
Configure DFDL, validation, and documentation properties.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

About this task

You can set properties for message, elements, types, groups, escape schemes, and formats. Different properties are displayed, depending on the object selected. The value of a property can be either inherited, or set locally. If the value is inherited, the following icon is shown to the left of the property value:  . For more information, see sections 10 - 17 of the DFDL v1.0 Specification.

Procedure

1. In the Outline view, select the appropriate message, element, type, group, escape scheme, or format.
   The selected object is displayed in the Editor area.
2. In the Editor area, select the element to configure.
   If properties can be configured for that element, they are displayed in the DFDL Properties area.
3. To change the type of an element, click the **Type** field in the Editor area.
   Other properties that are displayed in the Editor area can be set there.

4. Otherwise, find the property that you want to configure on the Representation Properties tab of the DFDL Properties area.

   These tools are available:

   - **Filter**, near the top of the Properties area.
   - The **Show Only Basic Properties** button at the top of the properties area will cause only basic properties to be displayed. If only basic properties are displayed, click **Show All Properties** to display all the properties.
   - The **Show Only Basic Properties** button at the top of the properties area hides properties that are inherited. If only local properties are displayed, click **Show Inherited Properties** to display all properties that this object possesses.
   - Hover over a property to see more information. If the property is inherited, information about the inheritance is displayed, and a link to the source is provided.

   Tip:

   - Use the property wizard  where available.
   - Use the **Validation** section to specify permissible values for the object. You can then check the data during test parsing.
   - Use the **Comment** property to add a brief note about an object.

Setting variables on a schema object
Set the value of a variable within a DFDL schema. You can use a variable to specify a value in your message model that is used later in parsing your message.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure
1. In the **Editor** view, select the DFDL schema object on which you want to set a variable.
2. On the **Variables** tab of the DFDL Properties area, in the **Set Variables** section, click **Add variable reference**.

   A list of available variables is displayed.
3. Select the variable from the list, and enter the required value for the variable in the **Value** field.
4. Save your message model.

Results
Your variable is used when the message model is run.

Adding an assert to a schema object
Add asserts to a DFDL schema object to define tests to ensure that data in the message are well-formed.

Before you begin
Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure
1. In the **Editor** view, select the DFDL schema object on which you want to add an assert.
2. On the **Asserts and Discriminators** tab of the DFDL Properties area, select **Asserts**, then click **Add assert**. If a discriminator is already applied to the object, the **Asserts** option is disabled, as an object cannot have both an assert and a discriminator applied to it.

3. Enter your test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.

   Content assistance is available for the **Test Condition** field; press Ctrl+Space to open the **XPath Expression Builder**.

Results

Your assert is used when the message model is run.

Adding a discriminator to a schema object

You can add a discriminator to a DFDL schema object to enable the DFDL parser to resolve a point of uncertainty.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the **Editor** view, select the DFDL schema object on which you want to add a discriminator.
2. On the **Asserts and Discriminators** tab of the DFDL Properties area, select **Discriminators**, then click **Add discriminator**. If an assert is already applied to the object, the **Discriminators** option is disabled, as an object cannot have both an assert and a discriminator applied to it.

   A new entry in the table is created.
3. Enter your test expression in the **Test Condition** field, and a human-readable failure message in the **Message** field.

   Content assistance is available for the **Test Condition** field; press Ctrl+Space to open the **XPath Expression Builder**.
4. Save your message model.

Results

Your discriminator is used when the message model is run.

Deleting objects

Delete an object that you have defined.

Before you begin

Create or open a DFDL schema file. See DFDL schema editor for a description of the DFDL schema editor.

Procedure

1. In the Outline view, select the object that you want to delete. If you do not see the object in the Outline view, click its parent object.
   For example, to rename a local element in a message, select the parent message.
2. In the editor area, click the object, and then click the **Delete** button.

Linking from one DFDL schema file to another

Add a link to a different schema.

Before you begin

1. Create two DFDL schema files.
2. Open the schema file that you want to add a link to.

About this task

Linking is needed when you have DFDL objects in one schema file, and you want to refer to them in a different schema file. For example, the first file contains a simple type, which an element in the second file wants to use.

Procedure

1. In the Outline area, right-click Schema and select **Add an import or include**.
2. Within the Add Schema Reference dialog that opens, browse to the schema that you want to include or import.

   The namespaces of the two schema files determine whether to use the `import` or `include` option.

	Target file has a target namespace	**Target file has notarget namespace**
Parent file has a target namespace	xsd:import	xsd:include[1]
Parent file has notarget namespace	xsd:import	xsd:include

[1] When a target namespace file includes a notarget namespace file, referencing an object in the target file from the parent file causes the object to be present in the namespace of the parent file.

Developing independent DFDL applications

Develop independent Java application programs that use the IBM DFDL API to parse a text or binary data stream into the equivalent logical information, or to serialize logical information into an equivalent text or binary data stream.

Before you begin

Before you start:

- Read about the IBM DFDL Java classes. Documentation for the IBM DFDL Java classes is provided in the following `.zip` file:

  install_dir`\server\tools\ibm-dfdl-java.zip`

  install_dir is the directory that you specified as the installation directory for IBM App Connect Enterprise.

About this task

A sample is provided to demonstrate how to use the IBM DFDL API. Run and explore the sample to learn what you can do with the IBM DFDL API. The IBM DFDL API sample is in the *install_dir*`\server\sample\dfdl` directory. For information about how to use the IBM DFDL API to develop independent programs, see “Configuring an environment for independent DFDL applications” on page 2240.

What to do next

- When you have written and tested your programs, you can distribute them to the systems that you want to run those programs. If a system that you want to run your IBM DFDL API independent programs does not have the IBM App Connect Enterprise runtime component installed, you must complete additional configuration steps, as described in “Configuring environments without the IBM App Connect Enterprise runtime component installed” on page 2241.

Configuring an environment for independent DFDL applications

Prepare the system environment in which you want to develop and run your independent applications that use the IBM DFDL API.

Before you begin

Before you start:

Before you can develop and run Java application programs that use the IBM DFDL API, you must install the following prerequisite software on your local computer environment:

- An IBM Java Development Kit (JDK) at a supported Java level.

  A JDK is not supplied with IBM App Connect Enterprise; you must acquire and install a suitable product yourself. You must use an IBM JDK to develop IBM DFDL independent programs and an IBM Java Runtime Environment (JRE) to run these programs. For information about supported Java levels, see IBM App Connect Enterprise system requirements.

About this task

To set up your computer in preparation for building and running IBM DFDL independent programs, you must configure your class path environment variable so that it includes all the JAR files needed for IBM DFDL. The instructions use the Windows path separator; but apply equally to other operating systems, if you change the separator.

Procedure

Update the class path environment variable:

1. Add all of the JAR files that are required for IBM DFDL.
   For example:

   ```
   set CLASSPATH=%CLASSPATH%;install_dir\server\dfdl\lib\ibm-dfdl.jar
   ```

   Repeat this step for each JAR file in your *install_dir*`\server\dfdl\lib` directory, where *install_dir* is the directory that you specified as the installation directory for IBM App Connect Enterprise. Alternatively, to add all of the required JAR files, run the `dfdlprofile` utility that is provided in your *install_dir*`\server\dfdl\bin` directory.
2. If you want to run and explore the IBM DFDL sample, add the `dfdlsample_java.jar` JAR file that is provided in your *install_dir*`\server\sample\dfdl` directory.
   For example:

   ```
   set CLASSPATH=%CLASSPATH%;install_dir\server\sample\dfdl\dfdlsample_java.jar
   ```
3. Add your Java development directory.
4. Add any other JAR files and directories that you require.

What to do next

Next:

- Use the tools that are provided by your JDK to build and run your IBM DFDL independent programs.
- If a system that you want to run your IBM DFDL API independent programs does not have the IBM App Connect Enterprise runtime component installed, you must complete additional configuration steps.

Configuring environments without the IBM App Connect Enterprise runtime component installed

Install and run Java independent programs that use the IBM DFDL API on systems that do not have the IBM App Connect Enterprise runtime component installed.

About this task

You can run Java programs that use the IBM DFDL API on systems where you have not installed the IBM App Connect Enterprise runtime component, but you must ensure that you have purchased a IBM App Connect Enterprise license for each computer on which you intend to run Java programs that use the IBM DFDL API.

The following instructions use the Windows path separator; but apply equally to other operating systems, if you change the separator.

Procedure

To install and run IBM DFDL independent programs in an environment that does not have the IBM App Connect Enterprise runtime component installed, complete the following steps:

1. Ensure that the target system has a compatible Java Runtime Environment (JRE) installed.

   Because you are not installing the IBM App Connect Enterprise runtime component, which includes a JRE, you must use an alternative option. For more information about Java support, see IBM App Connect Enterprise system requirements.
2. Copy the following files from a system that has the IBM App Connect Enterprise runtime component installed, to the target system:
   a) `ibm-dfdl-java.zip` from *`install_dir`*`\server\tools`, where *install_dir* is the directory that you specified as the installation directory for IBM App Connect Enterprise.
   b) Your Java independent program that uses the IBM DFDL API.
3. On the target system, extract the contents of `ibm-dfdl-java.zip` to a location of your choice; for example, `C:\dfdl`.

   Important: Extracting the contents of `ibm-dfdl-java.zip` installs license and entitlement files onto the target computer for use by IBM License Metric Tool (ILMT). You can use ILMT to monitor the usage of IBM (and other) software products.

   The extracted contents of `ibm-dfdl-java.zip` include a Software Identification (SWID) tag for `advanced` mode and a SWID tag for DFDL. The following SWID tags are detected by ILMT:

   - `ibm.com_IBM_App_Connect_Enterprise-12.0.3.swidtag`
   - `ibm.com_IBM_Data_Format_Description_Language_-_FREE-1.1.3.swidtag`

   If your IBM App Connect Enterprise installation is in `advanced` mode, no further action is needed. However, you must update the SWID tag in the directory where `ibm-dfdl-java.zip` is extracted so that ILMT reflects the correct license for the edition of IBM App Connect Enterprise that you have purchased if either of the following is true:

   - You have purchased a different edition of IBM App Connect Enterprise.
   - You change the operation mode to one of the following modes:
     - `standard`
     - `nonproduction`

   You can update the SWID tag by performing the following steps:

   a. On a machine on which you have installed IBM App Connect Enterprise, change to the `swidtag` directory to view the SWID tag files. The SWID tag file has the suffix `swidtag` if the operation mode is active or the suffix `deactivated` if the operation mode is inactive. If you are intending to change

the operation mode, see "Operation modes" on page 3 prior to changing the operation mode to ensure that the correct SWID tag is activated.

b. Copy the active SWID tag file from the `swidtag` directory to the directory where `ibm-dfdl-java.zip` is extracted:

- `ibm.com_IBM_App_Connect_Standard-12.0.3.swidtag` (`standard` mode)
- `ibm.com_IBM_App_Connect_Enterprise_for_NON-PRODUCTION-12.0.3.swidtag` (`nonproduction` mode)

c. Delete the `advanced` mode SWID tag, `ibm.com_IBM_App_Connect_Enterprise-12.0.3.swidtag`, from the directory where `ibm-dfdl-java.zip` is extracted.

For more information about ILMT, see the IBM License Metric Tool website

Update the class path environment variable:

4. Add all of the JAR files that are required for IBM DFDL.
   For example:

```
set CLASSPATH = %CLASSPATH%;C:\dfdl\lib\ibm-dfdl.jar
```

   Repeat this step for each JAR file in your *extract_dir*`\lib` directory, where *extract_dir* is the directory to which you extracted `ibm-dfdl-java.zip`. Alternatively, to add all of the required JAR files, run the `dfdlprofile` utility that is provided in your *extract_dir*`\bin` directory.
5. If you want to run and explore the IBM DFDL sample, add the `dfdlsample_java.jar` JAR file in your *extract_dir*`\samples` directory.
   For example:

```
set CLASSPATH = %CLASSPATH%;C:\dfdl\samples\dfdlsample_java.jar
```

6. Add your DFDL independent program JAR file.
7. Add any other JAR files and directories that you require.

Results

Next: You can now run your DFDL independent programs on the target computer.

Working with XML schema files

Create, open, modify, or delete an XML schema file.

About this task

This topic area describes the tasks that are involved in working with an XML schema file:

- "Opening an existing XML schema file" on page 2242
- "Creating an XML schema file" on page 2243
- "Deleting an XML schema file" on page 2244
- "Generating HTML documentation from an XML Schema file" on page 2245
- "Working with XML Schema message model objects" on page 2245

Opening an existing XML schema file

Open an existing XML schema file in the XML Schema editor to view or edit the contents of the file.

About this task

To open an existing XML schema file:

Procedure

1. Right-click the XML schema file (file extension `*.xsd`) that you want to open, and select **Open**.
   The XML Schema file opens in the XML Schema editor.

   Tip: The Eclipse framework lets you open resource files with other editors. However, you are advised to use the XML Schema editor to work with XML schema files. This editor correctly validates any changes that are made to your XML schema files. Other editors might not correctly validate files.
2. View or edit the data in your XML schema file.

Creating an XML schema file

You can create XML schema files to model your XML format messages

About this task

You can use XML schema to model various message formats, including:

- SOAP XML
- Other XML
- CORBA IDL
- SCA
- Database record

After you create an XML schema file, define the XML Schema elements needed to model your data.

You can create an XML schema file in the following ways:

- "Creating an XML schema file from scratch" on page 2243
- "Creating an XML schema file by using the New Message Model wizard" on page 2244

Creating an XML schema file from scratch

Create an XML schema file to model your XML format data.

Before you begin

Ensure that you have an application, library, or integration project in which to create your XML schema file. If the XML schema file will be used by multiple solutions, consider creating it in a shared library for ease of reuse.

About this task

You can also use a shared library to map between different versions of an XML schema file. Both versions might declare the same elements and types, which cannot coexist in one application. You can build one shared library that contains version 1 of an XML schema file, and another shared library that contains version 2. You can then build a map that references the XML schema files in the shared libraries and maps between the two versions. At deployment time, separate models are built for each of the shared libraries. Therefore, no clash results from the duplicated elements and types.

Procedure

To create an XML schema file, complete the following steps.

1. Click **File** > **New** > **Other**.
   A window opens in which you can select a wizard.
2. Expand **XML**, select **XML Schema File**, click **Next**.
   The **Create XML Schema** wizard opens.
3. Select a parent folder and enter a file name for your XML schema file.
4. Click **Finish**.

Results

Your XML schema file is displayed under the appropriate application, library, or integration project in the Application Development view, and is opened in the XML Schema editor.

What to do next

- Continue to edit your XML schema file by using the XML Schema editor.

Creating an XML schema file by using the New Message Model wizard

Use the **New Message Model** wizard to create XML schema files.

About this task

You can use the **New Message Model** wizard to create an XML schema file by any of the following methods:

- Create an empty XML schema file
- Create an XML schema file by importing an XML DTD file
- Create an XML schema file by using an XML document as an example
- Create an XML schema file by importing an existing schema file

To create an XML schema file:

Procedure

1. Click **File** > **New** > **Message Model ...**.
   The **New Message Model** wizard opens.
2. Select the message format that you want to model.
3. Step through the wizard, clicking **Next** to move to a new panel, then click **Finish** when you have entered the information that is required to create your XML schema file.

Results

Your new XML schema file is created, and is displayed in the **XML and DFDL Schemas** folder of your project.

What to do next

You can continue to edit your XML schema file by using the XML Schema editor.

Deleting an XML schema file

You can delete an XML schema file.

Before you begin

Create a file; see “Creating an XML schema file” on page 2243.

Procedure

1. Switch to the Integration Development perspective.
2. Right-click the XML schema file (file extension `.xsd`) that you want to delete, then click **Delete**.
   Alternatively, select the XML schema file that you want to delete, then, from the menu bar, click **Edit** > **Delete**, or press the Delete key.
3. In the **Confirm Resource Delete** window, click **Yes** to delete the XML schema file.
   Important: this action cannot be undone.

Results

Important: All files and objects that are associated with the XML schema file are deleted.

Generating HTML documentation from an XML Schema file

You can generate HTML documentation from an XML Schema file. The HTML documentation contains various information about the message model that is included in the XML Schema file. This documentation can be useful for providing a summary of the contents of your XML Schema file.

About this task

To generate HTML documentation from an XML Schema file:

Procedure

1. In the Navigator view, right-click the XML Schema file that you want to use to generate HTML documentation.
2. Click **Generate** > **HTML**. The **XSD Documentation Generation Options** dialog opens.
3. Select the style of HTML documentation that you want:
   - Select **Generate XSD Documentation with frames** to generate HTML files in a style that is similar to Javadoc files.
   - Select **Generate XSD Documentation without frames** to generate one HTML file per schema.
4. Specify a location for your HTML documentation. Click **Finish**.

Results

The HTML documentation is created in the location that you specified, and is visible in the Application Development view.

Working with XML Schema message model objects

Add, configure, and delete objects.

About this task

For information about how to add, configure, and delete XML Schema message model objects by using the XML Schema editor, see *Editing XML Schema properties* in the Rational Software Architect product documentation online.

Working with JSON schema files

Create, open, modify, or delete a JSON schema file.

About this task

This topic area describes the tasks that are involved in working with a JSON schema file:

- "Opening an existing JSON schema file" on page 2246
- "Creating a JSON schema file" on page 2246
- "Deleting a JSON schema file" on page 2247

Opening an existing JSON schema file

Open an existing JSON schema file in the JSON editor to view or edit the contents of the file.

About this task

A JSON schema file must either contain schema in the name (for example, `.schema.json`) or the first line of the file must indicate that it is a schema by including a `$schema` property. To open an existing JSON schema file:

Procedure

1. Right-click the JSON schema file that you want to open, and select **Open**.
   The JSON schema file opens in the JSON editor.

   Tip: The Eclipse framework lets you open resource files with other editors.
2. View or edit the data in your JSON schema file.

Creating a JSON schema file

Create a JSON schema file to model your JSON format data.

Before you begin

Ensure that you have an application, library, or integration project in which to create your JSON schema file. If the JSON schema file will be used by multiple solutions, consider creating it in a shared library for ease of reuse.

About this task

You can create a JSON schema file either from scratch in your IBM App Connect Enterprise Toolkit or by importing an existing JSON schema file into your IBM App Connect Enterprise Toolkit.

Procedure

- Create a JSON schema file from scratch:
  a) Click **File** > **New** > **Other**.
     A window opens in which you can select a wizard.
  b) Expand **General**, select **File**, click **Next**.
     The **Create New File** window opens.
  c) Select a parent folder and enter a file name for your JSON schema file.
     Give the file an extension of `.schema.json` or ensure that the file contains the `$schema` property.
  d) Click **Finish**.
- Import an existing JSON schema file from your file system.
  For information about importing files into the IBM App Connect Enterprise Toolkit from your file system, see “Importing files from the file system into the IBM App Connect Enterprise Toolkit” on page 2319.

Results

Your JSON schema file is displayed under the appropriate application, library, or integration project in the Application Development view, and is opened in the JSON editor.

What to do next

- Continue to edit your JSON schema file by using the JSON editor.

Deleting a JSON schema file

You can delete a JSON schema file.

Before you begin

Create a JSON schema file; see “Creating a JSON schema file” on page 2246.

Procedure

1. Right-click the JSON schema file that you want to delete. JSON schema files have the file extension `schema.json` or contain a `$schema` property at the root. Then click **Delete**.

   Alternatively, select the JSON schema file that you want to delete, then, from the menu bar, click **Edit** > **Delete**, or press the Delete key.
2. In the **Confirm Resource Delete** window, click **Yes** to delete the JSON schema file.

   Important: this action cannot be undone.

Results

Important: All files and objects that are associated with the JSON schema file are deleted.

Message Sets: Working with message sets

Use a message set to create a model for a particular message format.

About this task

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134.

The topics in this section describe how to use message sets and message definitions.

Enabling message set development

This task provides instructions for enabling message set creation and editing functionality in IBM App Connect Enterprise.

About this task

In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134.

You can import message flows containing message sets from previous versions, and they can be viewed, modified, and deployed in the usual way. However, by default, the menu options for creating new message sets or message definition files are hidden. To perform these tasks, you must first enable message set development in the IBM App Connect Enterprise Toolkit.

To enable message set development in IBM App Connect Enterprise, complete the following steps.

Procedure

1. From any perspective in the IBM App Connect Enterprise Toolkit, click **Window** > **Preferences**.
   The **Preferences** window opens.
2. Expand Integration Development, and select **Message sets**.
3. Under Message Set Development, select **Enable menus for Message Set development**.
4. Click **Apply**, then click **OK**.

Results

The **Preferences** window closes, and options for message set development are enabled in the IBM App Connect Enterprise Toolkit.

Message Sets: Working with a message set project

Creating and deleting a message set.

About this task

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134.

Before you begin to develop your message model, you must create a message set. A message set project is automatically created when you create a message set.

This topic area describes the tasks that are involved in working with a message set.

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Deleting a message set project” on page 2248

Message Sets: Deleting a message set project

Delete a message set project and, optionally, the contents of the associated project directory.

Before you begin

You must have completed the following task:

- “Message Sets: Creating a message set” on page 2250

Tip: Close all open windows within the IBM App Connect Enterprise Toolkit that relate to the message set project or associated files that you want to delete. If you do not do this, errors might occur when you try to process objects that no longer exist your workspace.

About this task

This task topic describes how to delete a message set project and, optionally, the contents of the associated project directory.

To delete a message set project:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message set project that you want to delete, then click **Delete** on the pop-up menu.

   The **Confirm Project Delete** window opens.
3. Choose whether to delete or retain the contents of the project directory.

   By default, project directory contents are not deleted. To delete the contents of the project directory, click **Also delete contents**; all files and directories that are associated with the project are deleted.
4. Click **Yes** to delete the message set project. Alternatively, click **No** or press **Esc** to cancel the deletion.

Results

Message Sets: Working with a message set

Complete a variety of tasks that are involved in working with a message set.

About this task

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

Message Sets: Configuring message set preferences

This task topic explains how to make changes to preferences that relate to message set processing. These preferences are for message set editors, message set model validation, and importing XML Schema.

About this task

To configure message set preferences:

Procedure

1. Open the **Preferences** window by clicking **Window** > **Preferences** > **Integration Development** > **Message Sets**.

   This displays the following options:

   - Editors
   - Validation
   - XML Schema Importer
2. View or make any necessary changes to the preferences for message set processing.

   These preferences are shown in the right hand area of the window.
3. When you have finished, click **Apply**. Alternatively, click **Restore Defaults** to return to the default settings for the displayed fields.
4. Close the **Preferences** window by clicking **OK**.

Results

Message Sets: Opening an existing message set

Open an existing message set in the Message Set editor so that you can view or edit its contents.

Before you begin

Create a message set by following the instructions in "Message Sets: Creating a message set" on page 2250.

Tip: Although you can open resource files with other editors you are advised to only use the IBM App Connect Enterprise Toolkit Message Set editor to work with message set files because this editor correctly validates changes made to the `messageSet.mset` files when they are saved. Other editors might not do this.

About this task

To open a message set so that you can view or edit its contents:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the `messageSet.mset` file of the message set that you are opening then click **Open** on the pop-up menu.

   This opens the Message Set editor for the selected file.

Results

You can now view or edit the file as required.

Message Sets: Creating a message set

Use the New Message Set wizard to create a message set.

About this task

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

The New Message Set wizard also creates a new message set project.

Note: You can also use a Quick Start wizard to create a message set, a message set project, and other resource files that you need to create a new application.

The New Message Set wizard allows you to select what kinds of message format you want to model in your message set. The message domain and the physical format created is inferred from the selection that you make. Note, however, that you can change the inferred domain using the message set editor.

The options are:

- XML documents (general)
- Web services (SOAP)
- Binary data (for example, C or COBOL structures)
- Text data (for example, CSV, SWIFT, or HL7)
- MIME documents other than web services
- Data for WebSphere Adapters
- Database records

The default value is `XML documents (general)`.

Below the list of message formats there are check boxes corresponding to each of the message formats. The check box corresponding to the message format that you selected is not available, but you can select any of the other check boxes to add other message formats to your message set.

If you later select a different default message domain, the checked state for the domain that you originally selected as the default does not change, but the check box is enabled.

As you can now select more than one message domain you can, for example, use the default value of **XML documents (general)** together with **Binary data (for example, C or COBOL structures)** and **Text data (for example, CSV, SWIFT or HL7)**. This results in the selection of the XMLNSC and MRM domains (to handle non-XML documents) within the same message set if you require this functionality.

The mapping between the selection, the domain, and the wire format created is described in the following table:

Selection	Inferred message domain	Physical format created
XML documents (general)	XMLNSC	XML
Web services (SOAP)	SOAP and XMLNSC	XML
Binary data (for example, C or COBOL structures)	MRM	CWF
Text data (for example, CSV, SWIFT, or HL7)	MRM	TDS
MIME documents other than web services	MIME	None
Data for WebSphere Adapters	DataObject	None
Data for CORBA	DataObject	None
Database records	DataObject	None

Depending on your selection, an appropriate IBM supplied message will be imported into the message set.

Note: The XML physical format is created only in case the user switches to MRM XML.

If you click **Finish** on the second page of the New Message Set wizard, the message set that is created has the following default property values:

Property	Default value
Message Domain	XML documents (general)
Physical Format	XML Wire Format (XML1)
Namespace support	Enabled

To create a new message set:

Procedure

1. Switch to the Integration Development perspective.
2. Click **New** > **Other** > **Message Set** to open the New Message Set wizard.
3. In the **Message set name** field, type the name for the message set that you are creating.

   The name that you type is also displayed in the **Message set project name** field.
4. Optional: You can choose a different message set project name; to do this, type this name into the **Message set project name** field.
5. Optional: You can specify a directory in which you want to store the project contents.

   If you do not specify a directory, the default workspace is used. To specify a directory, first clear the **Use default** check box, then either type into the **Directory** field the location of the directory, or click **Browse** to see a list of the folders that you can choose from for the location of the directory.
6. Optional: If you want to create a new message set whose definition is based on existing message set, click **Message Set** in the **Copy message set contents from another message set** pane and choose from the list of message set definitions that are shown; then click **Finish**.

   The new message set (and the message set project that contains it) is created immediately and the New Message Set wizard automatically closes.
7. Optional: If you want to create a message set whose definition is not based on an existing message set, click **Next**.

   You are presented with the next pane which allows you to choose the type of message data that you want to process.

   a) Expand the list shown under **Select the type of message data that you will be working with most often** and choose a value from the list shown. The value that you choose determines the default message domain of the message set. If you choose `XML Documents (general)`, the default message domain XMLNSC is used.

   b) Optional: You can now select additional types of message data. Under **Select any other types of message data that you will be working with** there are check boxes for each of the following message data types:

   - XML documents (general)
   - Web services SOAP
   - Binary data (for example, C or COBOL structures)
   - Text data (for example, CSV, SWIFT or HL7)
   - MIME documents other than web services
   - Data for WebSphere Adapters

   Note: These check boxes correspond to the list of data types from which you chose the data type that you will be working with most often, but the check box that corresponds to the data type that you chose from that list is not available.

   By default, all these check boxes are cleared. You can select any, or all of these check boxes, to add the corresponding data types to your message set.

   If you select the check box for text data, either for the type of message data that you will be working with most often or as another type of message data that you will be working with, you can additionally choose from the displayed list of text messaging standards. This list is the same as

that given in the description of the `Messaging Standard` property in Message Sets: TDS Format message set properties.

c) Click **Next**.

A new panel is displayed that summarizes some information about the message set that you have created. Specifically, it lists:

Supported message domains
Physical formats to be created
IBM supplied messages to be imported

8. Click **Finish** on this page to create the message set, and the message set project that contains it.

The New Message Set wizard closes.

Results

After the New Message Set wizard finishes, the message set editor is opened.

What to do next

You can now create some message definitions in the new message set. You can either create new message definitions from scratch, or create them based on existing artifacts such as WSDL, XSD, DTD, C, COBOL files, or EIS metadata. Use the Message Definition File wizard and the Message Definition File From wizard to help you with this.

Message Sets: Configuring logical properties: Message sets

Configure the logical properties of a message set using the Message Set editor.

Before you begin

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

If the `messageSet.mset` file for the appropriate message set is not already open in the Message Set editor, you must first open it as described in "Message Sets: Opening an existing message set" on page 2250.

About this task

This task topic describes how to configure the logical properties of a message set using the Message Set editor.

To configure the logical properties of a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, click **Message Set**.

   This displays the logical properties of the selected message set in the Details view.
3. Configure to your requirements the logical properties that are shown in the Details view.

   Note: Property fields that are disabled cannot be altered. For example, the **Message Set ID** field is disabled because the message set ID is generated automatically when the message set is created; the **Message Set ID** must not then be altered.
4. Save your changes by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Message Sets: Working with physical formats

If you are using the MRM domain to parse and write your messages, when you are developing your message model you might want to add one or more physical format layers to a message set, then configure the properties of these physical formats.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

About this task

This topic area covers the following tasks that relate to working with the physical properties of a message set:

- "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254
- "Message Sets: Configuring Custom Wire Format (CWF) properties: Message sets" on page 2255
- "Message Sets: Adding a TDS physical format" on page 2255
- "Message Sets: Configuring TDS properties: Message sets" on page 2256
- "Message Sets: Adding an XML wire format" on page 2256
- "Message Sets: Configuring XML Wire Format properties: Message sets" on page 2257
- "Message Sets: Renaming a physical format" on page 2258
- "Message Sets: Applying default physical format settings: Message sets" on page 2258
- "Message Sets: Removing a physical format" on page 2259
- "Message Sets: Observing 2007 U.S. changes to daylight saving time" on page 2259

Message Sets: Adding a Custom Wire Format (CWF)

You can add a Custom Wire Format (CWF) physical format layer to a message set by using the Message Set editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

About this task

To add a CWF physical format layer to a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the **Add Custom Wire Format** window by right-clicking **Custom Wire Formats**, then clicking **Add Custom Wire Format**.
3. On the **Add Custom Wire Format** window, specify the name that you want to use for the new CWF physical format. The default name is 'Binary1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new CWF physical format with 'CWF' or 'Binary', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.

4. Click **OK** to add the physical format layer to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Message Sets: Configuring Custom Wire Format (CWF) properties: Message sets
Configure the Custom Wire Format (CWF) properties of a message set using the Message Set editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250
- "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254

About this task

To configure the CWF properties of a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, the **Custom Wire Formats** node of the Properties Hierarchy shows the name of each of the CWF physical formats that have been added to the message set. If the physical format names are not in view, expand **Custom Wire Formats** by clicking **+**.
3. Click the chosen CWF physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the CWF properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Message Sets: Adding a TDS physical format
Add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

About this task

This task topic describes how to add a Tagged/Delimited String (TDS) physical format layer to a message set using the Message Set editor.

To add a TDS physical format layer to a message set:

Procedure

1. Switch to the Integration Development perspective.

2. In the Message Set editor, in the Properties Hierarchy, open the Add **Tagged/Delimited String Format** window by right-clicking **Tagged/Delimited String Formats** then clicking **Add Tagged/Delimited String Format** on the pop-up menu.
3. In the **Add Tagged/Delimited String Format** window, specify the name that you want to use for the new TDS format. The default name is 'Text1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new TDS physical format with 'TDS' or 'Text', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
4. Click **OK** to add the physical format to the message set. Alternatively, if you decide to cancel the process, click **Cancel** or press **Esc** on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the Message Set editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Message Sets: Configuring TDS properties: Message sets
Configure the Tagged/Delimited String (TDS) format properties of a message set using the Message Set editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250
- "Message Sets: Adding a TDS physical format" on page 2255

About this task

To configure the *TDS* physical format properties of a message set, take the following steps:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, the **Tagged/Delimited String Formats** node of the Properties Hierarchy shows the name of each of the TDS physical formats that have been added to the message set. If the physical format names are not in view, expand **Tagged/Delimited String Formats** by clicking **+**.
3. Click the chosen TDS physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the TDS properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Message Sets: Adding an XML wire format
You can add an XML wire format physical format layer to a message set by using the Message Set editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

About this task

This task topic describes how to add an XML wire format physical format layer to a message set using the Message Set editor.

To add an XML physical format layer to a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, open the **Add XML Wire Format** window by right-clicking **XML Wire Formats**, then clicking **Add XML Wire Format** on the pop-up menu.
3. On the **Add XML Wire Format** window, specify the name that you want to use for the new XML wire format. The default name is 'XML1'.

   Tip: Physical format names must be unique across a message set. You are recommended to start the name of your new XML physical format with 'XML', because this clearly identifies the type of the physical format that you are adding in relation to any of the other types.
4. Click **OK** to add the physical format layer. Alternatively, if you decide to cancel the process, click **Cancel** or press Esc on the keyboard.

   Tip: The new physical format layer is added with the relevant default property values. You can configure the message set properties according to your requirements, using the appropriate editor.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S.

Message Sets: Configuring XML Wire Format properties: Message sets
Configure the XML Wire Format properties of a message set using the Message Set editor.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Opening an existing message set” on page 2250
- “Message Sets: Adding an XML wire format” on page 2256

About this task

To configure the XML wire format properties of a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, the **XML Wire Formats** node of the Properties Hierarchy shows the name of each of the XML physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats** by clicking **+**.
3. Click the chosen XML physical format so that the properties of this format appear in the Details view of the Message Set editor.
4. Configure the XML wire format properties shown in the Details view according to your requirements.
5. Save your changes either by clicking **File> Save** or by pressing Ctrl+S. Alternatively click **File> Save All** or press Ctrl+Shift+S.

Results

Message Sets: Renaming a physical format
Rename a physical format using the Message Set editor.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Opening an existing message set” on page 2250

This task assumes that you have added one or more physical formats to the message set that you are working with. For further information see “Message Sets: Adding a Custom Wire Format (CWF)” on page 2254 or “Message Sets: Adding an XML wire format” on page 2256 or “Message Sets: Adding a TDS physical format” on page 2255.

About this task

This task topic describes how to rename a physical format using the Message Set editor.

To rename a physical format previously added to the message model:

Procedure

1. Close all message definition (`.mxsd`) files that are currently open in the Message Definition editor, otherwise you will not be able to rename the physical format.
2. Switch to the Integration Development perspective.
3. In the Message Set editor, the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited String Formats** by clicking **+**.
4. Right-click the physical format that you want to rename then click **Rename** on the pop-up menu to open the "**Rename wire format**" window.
5. In the "**Rename wire format**" window, type the new name for the physical format.

   The renaming operation modifies all of the message definition files in the message set and saves the amended message set file.
6. Click **Finish** to rename the physical format and save the message set file.

Results

Message Sets: Applying default physical format settings: Message sets
Apply the default settings to a physical format layer that has previously been added to a message set.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Opening an existing message set” on page 2250

The tasks in this topic area assume that you have added one or more physical formats to the relevant message set. For further information see “Message Sets: Adding a Custom Wire Format (CWF)” on page 2254 or “Message Sets: Adding an XML wire format” on page 2256 or “Message Sets: Adding a TDS physical format” on page 2255.

About this task

To apply the default settings to a physical format that has previously been added to a message set:

Procedure

1. Switch to the Integration Development perspective.
2. In the Message Set editor, in the Properties Hierarchy, right-click the physical format to which you want to apply the default settings then click **Apply default physical format settings** on the pop-up menu.

Results

The default settings are applied to the physical format that you have selected. No warning appears beforehand.

Message Sets: Removing a physical format
You can remove a physical format layer from your message set.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254 or "Message Sets: Adding an XML wire format" on page 2256 or "Message Sets: Adding a TDS physical format" on page 2255.

About this task

To remove a physical format layer from your message set:

Procedure

1. Close any message definition files that are currently open in the Message Definition editor, otherwise you will not be able to remove the physical format.
2. Switch to the Integration Development perspective.
3. In the Message Set editor, the Properties Hierarchy shows the name of each of the physical formats that have been added to the message set. If the physical format names are not in view, expand **XML Wire Formats**, **Custom Wire Formats**, or **Tagged/Delimited Wire Formats**, by clicking **+**.
4. Right-click the physical format that you want to remove, then click **Delete** on the pop-up menu.

   Tip: If you decide to proceed with deleting the physical format, all of the message definition files under the current message set are modified and the amended message set file is saved.
5. Click **Finish** to remove the physical format, or click **Cancel** to return to the Integration Development perspective without making any changes. Pressing Esc also returns you to the Integration Development perspective without making any changes.

Results

Message Sets: Observing 2007 U.S. changes to daylight saving time

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Opening an existing message set" on page 2250

This task assumes that you have added and configured one or more physical formats to existing message sets. For further information see: “Message Sets: Working with physical formats” on page 2254.

About this task

This task describes how to ensure that the message sets observe daylight saving time (DST) in line with the 2007 U.S. changes.

If your message sets use a named time zone that is *not* changing DST in line with the 2007 U.S. changes, you do not need to do anything.

If you are using a GMT-04:00, GMT-05:00, GMT-06:00, GMT-07:00, or GMT-08:00 named time zone with DST, that must observe DST in line with the 2007 U.S. changes, do the following steps on every computer on which the integration node is running:

1. Set the environment variable MQSI_USE_NEW_US_DST to an initial value: Y, for example.
2. Restart the integration node to use the changed DST.

Message Sets: Configuring documentation properties: Message sets

Document a message set in the IBM App Connect Enterprise Toolkit.

Before you begin

Complete the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Opening an existing message set” on page 2250

About this task

To configure the documentation for a message set:

Procedure

1. In the Message Set editor **Properties Hierarchy**, click **Message set**. The documentation text field appears in the Details view, with all the other logical properties of the message set.
2. Configure the documentation properties shown in the Details view to your requirements.

   You can use the `Documentation` property to set user-defined keywords and their value. These keywords are propagated to the integration node when you deploy the message set in a BAR file. These keywords are used to give additional information about the message set when you display deployed message set properties in the IBM App Connect Enterprise Toolkit. See “Message Sets: Message set version and keywords” on page 2173 for more information.
3. Save your changes by clicking **File** > **Save**, or by pressing Ctrl+S. Alternatively, click **File** > **Save All**, or press Ctrl+Shift+S.

Results

Message Sets: Deleting a message set

If you want to delete a message set from your message model, you must delete the message set project that contains the message set.

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message set project folder that contains the message set that you want to delete and click **Delete** on the pop-up menu.

   This opens the **Confirm Project Delete** window, which asks whether you want to delete the message set project that you have specified.

3. Click **Also delete contents .....** to delete the contents of the message set project, or click **Do not delete contents** to cancel the deletion of the message set project. Pressing the Esc key on your keyboard also cancels the deletion of the message set project.

Results

Important: When you delete a message set project, the action cannot be undone after you have confirmed the deletion. All folders and associated files for the message set project are deleted.

Message Sets: Working with a message definition file

Create, open, and delete a message definition file.

Before you begin

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134.

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

This topic area describes the tasks that are involved in working with a message definition file:

- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Deleting a message definition file" on page 2264

Message Sets: Opening an existing message definition file

This task topic describes how to open an existing message definition file in the Message Definition editor; you can then view and edit the contents of the file.

About this task

To open an existing message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message definition file (file extension `*.mxsd`) that you want to open, and select **Open**.

   This opens the Message Definition editor for the message definition file that you have specified.

   Tip: The Eclipse framework lets you open resource files with other editors. However, you are advised to use only the IBM App Connect Enterprise Toolkit Message Definition editor to work with message definition files, because this editor correctly validates any changes that are made to the message definition files. Other editors might not do this.
3. View or edit the data in the file as required.

Results

Message Sets: Creating a message definition file

Creating an empty message definition file to contain your message model objects.

Before you begin

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML Schema form.

You can create the message definition file in one of the following ways:

- Create the message definition file from scratch, see "Message Sets: Creating a message definition file from scratch" on page 2262.
- Base your message definition file on an existing resource (for example, an XML Schema file, an IBM supplied message, an XML DTD file, a C header file, a COBOL file, or a WSDL file), see "Message Sets: Creating a message definition file from an existing resource" on page 2263.
- Copy a message definition file from one message set to another.

  If you do copy a message definition file from one message set to another, you must check that the source and target message sets have identical physical formats, and that namespaces are enabled.

  If the source and target namespaces do not have identical formats, the physical format of the message definition file might be replaced by the default information applied to the target message set.

Message Sets: Creating a message definition file from scratch

Create an empty message definition file to contain message model objects.

Before you begin

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML Schema form.

To create an empty message definition file from scratch:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard.

   To do this, right-click the message set project in the Application Development view that you are adding the message definition file to, and click **New> Message Definition File** on the pop-up menu.

   The **Message Definition File** panel of the wizard is displayed. The target message set list is filtered to only show artifacts in the active working set.
3. Click the message set, type a name into the File name field, and click **Next**.
4. Step through the remainder of the wizard, completing the details as required.

Results

The new empty message definition file, with the name that you have specified and a file extension of `*.mxsd`, opens in the Message Definition editor; you can use the editor to create your own message

definitions. If you have chosen to use a target namespace, a directory structure that is based on the URI that you have supplied is created. The new message definition file is placed within this directory structure, which appears in the Application Development view.

Message Sets: Creating a message definition file from an existing resource
You must create a message definition file before you can create the message model objects. The message definition file contains the logical and physical model definitions of the objects in XML schema form.

Before you begin

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

To create a new message definition file that is based on an existing resource:

Procedure

1. Switch to the Integration Development perspective.
2. Open the appropriate New Message Definition File From wizard.

   Right-click the message set project in the Application Development view that you are adding the message definition file to, and click **New> Message Definition File From**.

   A submenu shows the list of resources from which you can choose.
3. Choose the resource on which to base your new message definition.

   Click one of the following resources:

   - **C Header File**
   - **COBOL File**
   - **CORBA IDL File**
   - **Database Definition File**
   - **IBM Supplied Message**
   - **SCA Import or Export**
   - **WSDL File**
   - **XML DTD File**
   - **XML Schema File**

   The first panel of the corresponding wizard is displayed.
4. Step through the remainder of the wizard supplying the details as required.

   For more information about using the New Message Definition File wizards, see Message Sets: New message definition file wizards.

Results

The new message definition file, with the name that you have specified and a file extension of `.mxsd`, opens in the Message Definition editor. You can use the editor to create your own message definitions. If you have chosen to use a target namespace, a directory structure that is based on the URI that you have supplied is created. The new message definition file is placed within this directory structure, which is shown in the Application Development view.

Message Sets: Deleting a message definition file

You can delete a message definition file from your message model.

About this task

To delete a message definition file from your message model:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message definition file (file extension `*.mxsd`) that you want to delete, then click **Delete**.

   Alternatively, select the message definition file that you want to delete in the Application Development view, then, from the menu bar, click **Edit** > **Delete**, or press the Delete key.
3. In the **Confirm Resource Delete** window, click **Yes** to delete the message definition file.

   Click **No**, or press the Esc key, to cancel the deletion of the message definition file.

Results

Important: All files and objects that are associated with the message definition file are deleted. This action cannot be undone.

Message Sets: Working with MRM message model objects

Add, configure, and delete objects.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262

About this task

This topic area describes the tasks that are involved in working with message model objects:

- “Message Sets: Adding MRM message model objects” on page 2264
- “Message Sets: Configuring MRM message model objects” on page 2277
- “Message Sets: Deleting objects” on page 2292

Message Sets: Adding MRM message model objects

Various tasks are involved in adding message model objects to a message definition file.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262

Before starting any of the tasks in this topic area, you must first open the message definition file to which you want to add message model objects in the Message Definition editor. See “Message Sets: Opening an existing message definition file” on page 2261 for further details.

About this task

This topic area describes the tasks that are involved in adding message model objects to a message definition file:

- "Message Sets: Adding a message" on page 2265
- "Message Sets: Adding a message from a global element" on page 2266
- "Message Sets: Adding a global element" on page 2267
- "Message Sets: Adding a local element" on page 2268
- "Message Sets: Adding an element reference" on page 2268
- "Message Sets: Adding a wildcard element" on page 2269
- "Message Sets: Adding a global attribute" on page 2270
- "Message Sets: Adding a local attribute" on page 2270
- "Message Sets: Adding an attribute reference" on page 2271
- "Message Sets: Adding a wildcard attribute" on page 2271
- "Message Sets: Adding a simple type" on page 2272
- "Message Sets: Adding a complex type" on page 2274
- "Message Sets: Adding a simple type to an element or attribute" on page 2281
- "Message Sets: Adding a complex type to an element" on page 2282
- "Message Sets: Adding a global group" on page 2274
- "Message Sets: Adding an attribute group" on page 2275
- "Message Sets: Adding a group reference" on page 2276

Message Sets: Adding a message
Add a message to a message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add a message to your message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message** on the pop-up menu.

   A simple message is immediately added to your message model and is assigned a default name.
4. Either type a new name for the message or press **Enter** to accept the default.

   Tip: When you add a message to your message model, an associated complex type and global element with the same name as the message are also created. The global element and the message cannot have different names and changing the name of one changes the names of both. The complex type can be renamed.

Results

You can now configure the properties of the message to your exact requirements. For further information about configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a message from a global element
Add a message from a global element to a message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding a global element" on page 2267 (This must be a global element of complex type)

About this task

To add a message from a global element to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message From Global Element** on the pop-up menu to open the **Add Message From Global Element** window.

   This window lists all the global elements of a complex type that are not already associated with a message.
4. In the **Select a global element of complex type that is not already used for a message** list, click the global element that you want to use to create your message.
5. Click **OK**.

   This immediately adds a message with the same name as the selected global element to your message model.

Results

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a message from a global type
Add a message from a global type to your message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

- "Message Sets: Adding a global element" on page 2267 (This must be a global element of complex type)

About this task

To add a message from a global type to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Messages** then click **Add Message From Global Type** on the pop-up menu to open the **Add Message From Global Type** window.

   This window lists all the global complex types that are not already associated with a message.
4. In the **Select a global complex type** list, click the global complex type that you want to use to create your message.
5. Click **OK**.

   This immediately adds a message with the same name as the selected global complex type to your message model.

Results

You can now configure the properties of the message to your exact requirements. For further information on configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a global element
Add a global element to a message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add a global element to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Elements and Attributes** then click **Add Global Element** on the pop-up menu.

   This adds a global element of type string to your message model, and assigns a default name.
4. Either type a new name for the global element or press Enter to accept the default.

Results

You can now configure the global element to your requirements. For further information on configuring message model objects see “Message Sets: Configuring MRM message model objects” on page 2277.

Message Sets: Adding a local element
Add a local element to a message, type, group, or complex element.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

This task assumes that you have previously added the relevant message, type, group, or complex element to your message model.

About this task

To add a local element to a message, type, group, or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding a local element then click **Add Local Element** on the pop-up menu.

   A local element of type string is added to the message model and is assigned a default name.
4. Either type a new name for the local element or press **Enter** to accept the default.

Results

You can now configure the local element to your exact requirements. For further information about configuring message model objects see “Message Sets: Configuring MRM message model objects” on page 2277.

Message Sets: Adding an element reference
Add an element reference to a message, type, group, or complex element.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

This task assumes that you have previously added the relevant message, type, global group, or complex element to your message model.

About this task

To add an element reference to a message, type, global group, or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding the element reference, then click **Add Element Reference** on the pop-up menu.

   This adds a default element reference to the message model object that points to an existing global element. This existing global element might be in the current message definition file or in a message definition file defined under **Includes** or **Imports** for the current message definition file. For further information about Imports and Includes, see "Message Sets: Linking from one message definition file to another" on page 2293.

Results

You can now configure the element reference to your exact requirements. For further information about configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a wildcard element
Add a wildcard element to a message, type, group, or complex element in a message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

You can add a wildcard element to a message, type, group or complex element. This task assumes that you have previously added the relevant message, type, group or complex element to your message model.

About this task

To add a wildcard element to a message, type, group or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the message model object (message, complex type, group or complex element) to which you are adding the wildcard element then click **Add Wildcard Element** on the pop-up menu.

   A wildcard element is added and appears in the Outline view.

Results

You can now configure the wildcard element to your exact requirements. For further information on configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a global attribute
Add a global attribute to your message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add a global attribute to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Elements and attributes** then click **Add Global Attribute** on the pop-up menu.

   A global attribute of type string is immediately added and is assigned a default name.
4. Either type a new name for the global attribute or press Enter to accept the default.

Results

You can now configure the global attribute to your requirements. For more information on configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a local attribute
Add a local attribute to a message, complex type, or complex element.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

You can add a local attribute to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add a local attribute to a message, complex type or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.

3. In the Outline view, right-click the object (message, complex type, complex element, or attribute group) to which you are adding the local attribute then click **Add Local Attribute** on the pop-up menu.

   A local attribute of type string is immediately added to the message model object and is assigned a default name.
4. Either type a new name for the local attribute or press **Enter** to accept the default.

Results

You can now configure the local attribute to your requirements. For further information about configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding an attribute reference
Add an attribute reference to a message, complex type, or complex element.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

You can add an attribute reference to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add an attribute reference to a message, complex type or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the message model object (message, complex type, complex element, or attribute group) to which you are adding the attribute reference then click **Add Attribute Reference** on the pop-up menu.

   This action adds a default attribute reference to the message model object that points to an existing global attribute.

Results

You can now configure the attribute reference to your exact requirements. For further information about configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a wildcard attribute
Add a wildcard attribute to a message, complex type, or complex element.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262

- “Message Sets: Opening an existing message definition file” on page 2261

You can add a wildcard attribute to a message, complex type, or complex element. This task assumes that you have previously added the relevant message, complex type, or complex element to your message model.

About this task

To add a wildcard attribute to a message, complex type or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, complex element, or attribute group) to which you are adding the wildcard attribute then click **Add Wildcard Attribute** on the pop-up menu.

   A wildcard attribute of type string is immediately added to the message model object and is assigned a default name.
4. Either type a new name for the wildcard attribute or press **Enter** to accept the default.

Results

You can now configure the wildcard attribute to your requirements. For further information about configuring message model objects see “Message Sets: Configuring MRM message model objects” on page 2277.

Message Sets: Adding a simple type
Add a simple type to your message model.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

About this task

To add a simple type to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Types** then click either **Add Simple Type Restriction**, **Add Simple Type List**, or **Add Simple Type Union** on the pop-up menu.
   - For a restriction, a simple type of base type string is added, and assigned a default name.
   - For a list, a simple type of item type string is added, and assigned a default name.
   - For a union, a simple type with a single member type of string is added, and assigned a default name.

4. Either type a new name for the simple type or press **Enter** to accept the default.

Results

You can now configure the simple type to your exact requirements.

If the simple type is a restriction:

- You can change the base type using the editor Properties view.
- You can set the value constraints associated with the simple type. See "Message Sets: Setting value constraints" on page 2283.
- You can replace the base type with a new local simple type. In the Outline view right-click simple type and click one of:
  - **Add Simple Type Restriction**. This option replaces the base type with a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction'. The result is that the original simple type becomes a restriction of a restriction.
  - **Add Simple Type List**. This option replaces the base type with a new simple type list, with an item type of string. You can configure the list as described in 'If the simple type is a list'. The result is that the original simple type becomes a restriction of a list. It appears as a list in the editor, because a restriction of a list is itself a list, but you can also set certain value constraints.

If the simple type is a list:

- You can change the item type using the editor Properties view.
- You can replace the item type with a new local simple type. In the Outline view right-click the simple type and click one of:
  - **Add Simple Type Restriction**. This option replaces the item type with a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction'. The result is that the original simple type becomes a list of a restriction.
  - **Add Simple Type Union**. This option replaces the item type with a new simple type union, with a single member type of string. You can configure the union as described in 'If the simple type is a union'. The result is that the original simple type becomes a list of a union.

If the simple type is a union:

- If the member type of string is not required, in the Outline view right-click the string and click Delete.
- You can add further members to the union. In the Outline view right-click the simple type and click one of:
  - **Add Union Member Type**. This option adds a union member that is an existing simple type. Use the type selection dialog to select the simple type required.
  - **Add Local Member Type Restriction**. This option adds a union member that is a new simple type restriction, with a base type of string. You can configure the restriction as described in 'If the simple type is a restriction' referred to earlier.
  - **Add Local Member Type List**. This option adds a union member that is a new simple type list, with an item type of string. You can configure the list as described in 'If the simple type is a list' referred to earlier.
  - **Add Local Member Type Union**. This option adds a union member that is a new simple type union, with a single member type of string. You can configure the new union as described in 'If the simple type is a union'.
- New members are added to the end of the union. To change the order of a member, in the Outline view select the member and drag it to the required position in the union. All union members that are existing simple types must occur ahead of all members that are local restrictions, lists, and unions, so reordering is subject to this rule.

For further information about configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a complex type
Add a complex type to your message model.

Before you begin

You must already have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add a complex type to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Types** then click **Add Complex Type** on the pop-up menu.
   A complex type is added and is assigned a default name.
4. Either type a new name for the complex type or press Enter to accept the default.

Results

You can now configure the complex type to your requirements. For further information on configuring message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a global group
Add a global group to your message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add a global group to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Group** on the pop-up menu.
   A global group is added to your message model and is assigned a default name.
4. Either type a new name for the global group or press **Enter** to accept the default.

Results

You can now configure the global group to your requirements. For further information about configuring the properties of message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a local group
Add a local group to a message, type, global group, or complex element.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

You can add a local group to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

About this task

To add a local group to a message, complex type, group, or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you are adding the local group then click **Add Local Group** on the pop-up menu.

   A local group is immediately added to the message model with type composition set to *sequence* by default.

Results

You can now configure the local group to your requirements. For further information about configuring message model objects, see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding an attribute group
Add an attribute group to your message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To add an attribute group to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group** on the pop-up menu.

   A global attribute group is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group or press **Enter** to accept the default.

Results

You can now configure the attribute group to your requirements. For further information about configuring the properties of message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding a group reference
You can add a group reference to a message, type, global group, or complex element.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

You can add a group reference to a message, complex type, group, or complex element. This task assumes that you have previously added the relevant message, complex type, group, or complex element to your message model.

About this task

To add a group reference to a message, complex type, group or complex element:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the object (message, complex type, group, or complex element) to which you want to add a group reference then click **Add Group Reference** on the pop-up menu.

   A group reference is immediately added to your message model.

Results

You can now configure the group reference to your requirements. For further information on configuring the properties of message model objects see "Message Sets: Configuring MRM message model objects" on page 2277.

Message Sets: Adding an attribute group
Add an attribute group reference to a message model.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

About this task

To add an attribute group to your message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click **Groups** then click **Add Attribute Group** on the pop-up menu.
   An attribute group is added to your message model and is assigned a default name.
4. Either type a new name for the attribute group reference or press Enter to accept the default.

Results

You can now configure the attribute group to your requirements using the Message Definition editor. For further information on configuring the properties of message model objects see “Message Sets: Configuring MRM message model objects” on page 2277.

Message Sets: Configuring MRM message model objects

Various tasks are involved in configuring MRM message model objects

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Adding MRM message model objects” on page 2264 (You must have added one or more objects to your message model)

Before starting any of the tasks in this topic area, you must first open the message definition file for which you want to configure message model objects in the Message Definition editor. See “Message Sets: Opening an existing message definition file” on page 2261 for further details.

About this task

This topic area describes the tasks that are involved in configuring message model objects:

- “Message Sets: Renaming objects” on page 2278
- “Message Sets: Reordering objects” on page 2278
- “Message Sets: Copying objects” on page 2279
- “Message Sets: Pasting objects” on page 2280
- “Message Sets: Changing the type of an element or attribute” on page 2280
- “Message Sets: Setting value constraints” on page 2283
- “Message Sets: Configuring logical properties: Message model objects” on page 2284
- “Message Sets: Configuring documentation properties: Message model objects” on page 2286

- "Message Sets: Configuring physical properties" on page 2287
  - "Message Sets: Configuring Custom Wire Format (CWF) properties: Message model objects" on page 2287
  - "Message Sets: Configuring XML Wire Format properties: Message model objects" on page 2290
  - "Message Sets: Configuring TDS properties: Message model objects" on page 2288
  - "Message Sets: Applying default physical format settings: Message model objects" on page 2291
- "Message Sets: Deleting objects" on page 2292

Message Sets: Renaming objects
You can rename message model objects in the IBM App Connect Enterprise Toolkit.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

Objects in the IBM App Connect Enterprise Toolkit such as files, messages and elements can have different physical representations. Eclipse handles renaming differently depending on the object.

Tip: Not all objects can be renamed. For example, you cannot rename wildcards, local groups, or local types, because they do not have a name.

If an object can be renamed the usual way to do it is as follows:

Procedure

1. Switch to the Integration Development perspective.
2. In the Outline view, right-click the object that you want to rename then click **Rename** on the pop-up menu.

   Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Rename** on the pop-up menu.

   In both cases, depending on the object, either a renaming dialog opens or you will now be able to edit the name of the object directly.
3. Type the new name for the object.
4. If the renaming dialog is open, either press Enter or click **OK**.

Results

Message Sets: Reordering objects
Reorder message model objects within a message definition file.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

- “Message Sets: Adding MRM message model objects” on page 2264 (You must have added one or more objects to your message model)

About this task

To reorder objects within a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. Click the object that you want to move.
   For example, you could select a local element within a message in either the Outline view or Properties Hierarchy.
3. Use the mouse to drag the object to its new location.

   Tip: As you drag the object and the mouse cursor passes between objects, a black line appears, showing where the current focus is. If you try to drag the object to a location that it cannot be moved to (including objects that are highlighted as the cursor passes over them), the object remains in its original position when you release it.

Results

Message Sets: Copying objects
You can copy an object in a message definition file, such as a message for a local element object, or types for a complex type object.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261
- “Message Sets: Adding MRM message model objects” on page 2264 (You must have added one or more objects to your message model)

About this task

To copy an object:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, right-click the message model object that you want to copy, then click **Copy**.
   Alternatively, right-click the object in the Message Definition editor **Overview** tab, then click **Copy**.

Results

The object is now copied.

Message Sets: Pasting objects
Paste objects that you have previously copied within the same message definition file

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)
- "Message Sets: Copying objects" on page 2279

About this task

You can paste objects that you have previously copied within the same message definition file.

You can only copy and paste an object within the same message definition. You cannot copy an object and paste it into another message definition, either within the same message set or in a different message set.

To paste an object in the message definition from which you copied it:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, right-click the location where you are going to paste the object then click **Paste** on the pop-up menu.

   Alternatively, right-click the object in the Message Definition editor **Overview** tab then click **Paste** on the pop-up menu.

   The object appears in the new location with a default name which you can change if you want to.
4. Either type a new name for the object or press **Enter** to accept the default.

What to do next

Note: When you copy and paste objects within the message set, where physical properties exist for that object, these settings are not pasted, but are set to default values.

Tip: If you cannot select **Paste** from either menu, you might be trying to paste to a location that is not valid. For example, you cannot paste a complex type into a local element.

Message Sets: Changing the type of an element or attribute
You can change the type to a local element, global element, local attribute, or global attribute.

Before you begin

You must already completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

You can change the type of an element or attribute in your message model to another existing type, or you can create a new simple type or a new complex type.

To change the type of an element or attribute to an existing type:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, click the element for which you want to change the type.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy click **Logical Properties** > **Global Element** (or **Logical Properties** > **Local Element**, **Logical Properties** > **Global Attribute**, or **Logical Properties** > **Local Attribute**).

   If necessary, expand **Logical Properties** by clicking **+**.
6. In the Details view, in the **Type** property, click the new type that you require.

   Tip: If the type you require is not displayed, you can find it by clicking **(More...)** in the list. This displays the **Type Selection** window with additional options. If you know which type you require, specify the first letter in the text box at the top of the **Type Selection** window. Matching types are then displayed, making the selection process easier.
7. When you have selected the type that you require, click **OK**.

Results

The change to the type is applied wherever the element or attribute occurs.

The task above explains how to switch to an existing type. If you want to create a new simple type or a new complex type, select **(New Simple Type Restriction)**, **(New Simple Type List)**, **(New Simple Type Union)**, or **(New Complex Type)** in the **Type** list (see step 6 above). For information on how to create a new simple type or a new complex type see “Message Sets: Adding a simple type to an element or attribute” on page 2281 or “Message Sets: Adding a complex type to an element” on page 2282.

Message Sets: Adding a simple type to an element or attribute
You can add a simple type to a local element, global element, local attribute, or global attribute.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

This task assumes that you have previously added the relevant element or attribute to your message model.

About this task

To add a new simple type:

Procedure

1. Switch to the Integration Development perspective.

2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the element to which you want to add a new simple type.
4. In the Message Definition editor, click the **Properties** tab.
5. In the Properties Hierarchy, click **Logical Properties > Global Element** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
6. In the Details view, in the **Type** property, select **(New Simple Type Restriction)**, **(New Simple Type List)**, or **(New Simple Type Union)** to open the relevant **New Simple Type** window to create a simple type of the type that you specify.
7. In the **New Simple Type** window, in the **Base Type** list, click the type that you want to use.
8. Optional: If you want to add the new simple type as a global simple type, select the **Create as Global Simple Type** check box and specify the name for your new simple type in the **Name** field.
9. Click **OK**.

   A simple type is immediately added to your message model.

Results

Any changes that you make are reflected throughout where the element to which you have added a new simple type occurs.

Message Sets: Adding a complex type to an element
You can add a complex type to a local element, global element, local attribute, or global attribute.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

This task assumes that you have previously added the relevant element or attribute to your message model.

About this task

When you add a complex type to an element or attribute, you can either create a new complex type or derive a new complex type from an existing base type.

To add a new complex type:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit.

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the element to which you want to add a new complex type.
4. In the Message Definition editor, click the **Properties** tab.
5. In the Properties Hierarchy, click **Logical Properties > Global Element** (or **Logical Properties > Local ElementLogical Properties > Global Attribute**, or **Logical Properties > Local Attribute**).
6. In the Details view, in the **Type** list, click **(New Complex Type)** to display the **New Complex Type** window.

7. If you want to create a new local complex type, click **Create a Local Complex Type**, in the **Composition** list, click the option that you require.
8. If you want to derive a new local complex type from an existing base type:
   a) Click **Derive a new Local Complex Type from existing base type**.
   b) In the **Base Type** list, click the base type that you want to use.
      Depending on the base type you select, the **Composition** and **Derived By** lists might become active.
   c) If the **Composition** and **Derived By** lists are active, click the options that you require in both lists.
9. If you want to add the new complex type as a global complex type, select the **Create as Global Complex Type** check box, and specify a name for your new complex type in the **Name** field.
10. Click **OK** to close the **New Complex Type** window and add the new complex type to your message model.

Results

Any changes that you make are reflected throughout where the element to which you are adding the complex type occurs.

Message Sets: Setting value constraints
You can set the value constraints associated with a simple type.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding a simple type" on page 2272 or "Message Sets: Adding a simple type to an element or attribute" on page 2281 (You must have added one or more global or local simple types to your message model)

About this task

Value constraints are usually associated with a simple type; they refine a simple type by defining limits on the values which the simple type can represent. To set the value constraints associated with a simple type:

Procedure

1. Switch to the Integration Development perspective.
2. In the Outline view, click the simple type you are updating.
   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**.
3. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the lower-left corner of the editor area.
   The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation

4. In the Properties Hierarchy under **Logical Properties** click **Value Constraints**.
   This displays the current value constraints settings for the selected simple type in the Details pane.
   Tip: If **Value Constraints** is not in view, expand **Logical Properties** by clicking **+**.
5. Set the value constraints for the selected simple type by making the appropriate changes to the information shown in the Details pane.

Results

Setting an enumeration

About this task

An enumeration restricts which values can be set for the value constraint. For example, "ABC" and "123". Use this section to create a list of fixed values that the associated type must match.

To set an enumeration:

Procedure

1. Click **Add** to the right of the **Enumerations** field.
   This adds an enumeration that has a default enumeration (for example *enumeration1*).
2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each enumeration that you are adding.

Setting a pattern

About this task

Set a pattern to indicate that the value constraint defines a string used as a regular expression that must be matched by the data in the associated type. The regular expression syntax supported is XML Schema regular expressions.

See Message Sets: Regular expression syntax for a list of supported regular expression syntax elements.

To set a pattern:

Procedure

1. Select **Add** to the right of the **Patterns** field.
   This adds a pattern that has a default pattern (for example *pattern1*) and is in update mode.
2. Type the data that you want to set for this value constraint.
3. Press Enter on your keyboard.
4. Repeat the above steps for each pattern that you are adding.

Message Sets: Configuring logical properties: Message model objects
You can configure the logical properties of an object that has previously been added to the message model.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262
- “Message Sets: Opening an existing message definition file” on page 2261

- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

To configure the logical properties of an object that is part of the message model:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**. If the information listed above is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the message model object for which you want to configure the logical properties:
   a) Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** as appropriate by clicking **+**.
   b) Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

   The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation

   The type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

   If the items under **Logical Properties** are not in view, expand **Logical Properties** by clicking **+**.
5. Display the logical properties of the selected object in the Details view of the Message Definition editor, by clicking the appropriate item under **Logical Properties**.
6. Configure the logical properties of the selected item to your requirements by making the appropriate changes to the information shown in the Details view.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or by pressing Ctrl+S. Alternatively click **FileSave All** or press Ctrl+Shift+S.

Results

Message Sets: Configuring documentation properties: Message model objects
You can configure the documentation properties of an object that has previously been added to the message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

To configure the documentation properties of an object contained within a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**. If the information listed above is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the message model object for which you want to configure the documentation properties by taking the following steps:
   a) Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** as appropriate by clicking **+**.
   b) Click the object you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

   The Properties Hierarchy displays the following nodes:
   - Logical Properties
   - Physical Properties
   - Documentation

   The type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each of these nodes.

   Tip: If the items under **Documentation** are not in view, expand **Documentation** by clicking **+**.
5. Display the logical properties of the selected object in the Details view by clicking the appropriate item under **Documentation**.

6. Configure the documentation properties of the selected item to your requirements by typing text into the **Documentation** field.

   Right-clicking in the field allows you to select options for undoing changes you have made, cutting or copying text from the field, pasting text into the field, deleting highlighted text or selecting all text in the field.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** from the menu or pressing Ctrl+S. Alternatively, from the menu, click **File** > **Save All** or press Ctrl+Shift+S.

Results

Message Sets: Configuring physical properties
Working with the physical properties of message model objects.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

The tasks in this topic area assume that you have added one or more physical formats to a message set. For further information see "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254 or "Message Sets: Adding an XML wire format" on page 2256 or "Message Sets: Adding a TDS physical format" on page 2255.

About this task

When you have added objects to your message model it is likely that you will want to configure the physical properties of these objects. The following tasks relate to configuring the physical properties of message model objects:

- "Message Sets: Configuring Custom Wire Format (CWF) properties: Message model objects" on page 2287
- "Message Sets: Configuring XML Wire Format properties: Message model objects" on page 2290
- "Message Sets: Configuring TDS properties: Message model objects" on page 2288
- "Message Sets: Applying default physical format settings: Message model objects" on page 2291

Message Sets: Configuring Custom Wire Format (CWF) properties: Message model objects
You can configure the Custom Wire Format (CWF) properties of a message model object by using the Message Definition editor

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

To configure the *CWF* properties of a message model object:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the CWF properties by doing the following.
   a) Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b) Click the object you that want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

   In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

   Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the CWF physical format is called Binary1 but could have a user defined name instead.
5. Under **Physical Properties**, click the object type for the message model object that you have chosen to configure under the CWF physical format.

   The CWF properties of your selected message model object appear in the Details view.
6. Configure the CWF properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.
7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or pressing Ctrl+S. Alternatively click **FileSave All** or press Ctrl+Shift+S.

Results

Message Sets: Configuring TDS properties: Message model objects
You can configure the Tagged/Delimited String (TDS) properties of a message model object.

Before you begin

You must have completed the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Message Sets: Creating a message definition file” on page 2262

- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding a TDS physical format" on page 2255
- "Message Sets: Adding MRM message model objects" on page 2264 (Adding one or more objects to your message model)

About this task

To configure the *TDS* properties of a message model object:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the TDS properties by taking the following steps:
   a) Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b) Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

   In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

   Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the TDS physical format is called Text1 but could have a user defined name instead.
5. Select the **Properties** tab located in the lower-left corner of the Message Definition editor.
6. Under **Physical Properties**, under the TDS physical format, click the object type for the message model object that you have chosen to configure.

   The TDS physical format properties of your selected message model object appear in the Details view.
7. Configure the TDS physical format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.
8. Save your changes by selecting **File** > **Save message_definition_file.mxsd** from the menu or press Ctrl+S. Alternatively, from the menu, select **File** > **Save All**, or press Ctrl+Shift+S.

Results

Message Sets: Configuring XML Wire Format properties: Message model objects
You can configure the XML Wire Format properties of a message model object.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding an XML wire format" on page 2256
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

To configure the *XML Wire Format* properties of a message model object:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit and is displaying the following information:
   - The name of the message definition file
   - Messages
   - Types
   - Groups
   - Elements and Attributes

   If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**. If the above hierarchy is not displayed, ensure that the message definition file is open in the Message Definition Editor. Message definition files have an `.mxsd` file extension.
3. In the Outline view, select the object for which you want to configure the XML Wire Format properties by taking the following steps:
   a) Depending on the type of the object that you are selecting, expand **Messages**, **Types**, **Groups** or **Elements and Attributes** by clicking **+**.
   b) Click the object that you want to select within the expanded node.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.

   In the Message Definition editor, in the Properties Hierarchy, the name of each of the physical formats that have been added to the message set appears under **Physical Properties**. The object type (for example, Local Element or Global Element) of the message model object that you selected in the Outline view is displayed under each physical format shown.

   Tip: If the physical formats are not in view in the Properties Hierarchy, expand **Physical Properties** by clicking **+**. By default the XML Wire Format is called XML1 but could have a user defined name instead.
5. Under **Physical Properties**, under the XML Wire Format, click the object type for the message model object that you have chosen to configure.

   The XML Wire Format properties of your selected message model object appear in the Details view of the Message Definition editor.
6. Configure the XML Wire Format properties of the selected object to your requirements by making the appropriate changes to the information shown in the Details view.

   Note: It is not possible to configure disabled fields.

7. Save your changes by clicking **File** > **Save message_definition_file.mxsd** or pressing Ctrl+S. Alternatively select **File** > **Save All** from the menu or press Ctrl+Shift+S.

Results

Message Sets: Applying default physical format settings: Message model objects
You can apply the default physical format settings to a message model object that is contained in a message definition file.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

This task assumes that you have added one or more physical formats to the relevant message set. For further information see "Message Sets: Adding a Custom Wire Format (CWF)" on page 2254 or "Message Sets: Adding an XML wire format" on page 2256 or "Message Sets: Adding a TDS physical format" on page 2255.

About this task

To apply the default physical format setting to a message model object previously added to a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. In the Outline view, click the object to which you want to apply default physical format settings.
3. Click the **Properties** tab located in the lower-left corner of the Message Definition editor.
4. Check that the Message Definition editor Properties Hierarchy displays the following information:
   - Logical Properties
   - Physical Properties (For each of the physical formats that have been added to the message set, the name of the physical format appears under **Physical Properties**. Under each physical format the type of message model object that you selected is displayed as a child.)
   - Documentation

   Ensure that **Physical Properties** in the Properties Hierarchy is fully expanded by clicking **+**.
5. Right-click on the message model object type underneath the physical format to which you want to apply the default settings then click **Apply default physical format settings**.

   The default physical format settings for the message model object that you selected are applied without warning.
6. Save your changes by clicking **File** > **Save message_definition_file.mxsd** from the menu or pressing Ctrl+S. Alternatively, from the menu, click **File** > **Save All**, or press Ctrl+Shift+S.

Results

Message Sets: Deleting objects

Delete an object from your message model.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261
- "Message Sets: Adding MRM message model objects" on page 2264 (You must have added one or more objects to your message model)

About this task

To *remove* objects contained within a message definition file:

Procedure

1. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
2. In the Outline view, right-click the object that you want to remove then click **Delete** on the pop-up menu.

   Alternatively right-click the object in the Message Definition editor **Overview** tab, and then click **Delete**.

   The type of object and the relationship that it has with other objects determines whether the object is now deleted without a confirmation window appearing, or whether a confirmation window opens with a list of all the objects that will be deleted along with the one that you have selected.
3. If a confirmation window opens, click **OK** to delete the objects.

   Tip: You can undo a deletion by selecting **Edit> Undo**, provided that you have not saved the changes that you have made.

Results

Message Sets: Creating a multipart message

A multipart message occurs when you embed a message in another message.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

To create a multipart (embedded) message:

Procedure

1. Switch to the Integration Development perspective.

2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, add one of the following objects to your message model:
   - A complex type (for further information on completing this task see "Message Sets: Adding a complex type" on page 2274)
   - A global group (for further information on completing this task see "Message Sets: Adding a global group" on page 2274)
   - A local group (for further information on completing this task see "Message Sets: Adding a local group" on page 2275)

   Tip: You can also use a local ANONYMOUS complex type when creating a multipart message. For further information see "Message Sets: Adding a complex type to an element" on page 2282.
4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy, under **Logical properties**, click one of the following items, depending on which of these you added in step 3:
   - **Complex Type**
   - **Global Group**
   - **Local Group**
6. In the Details view, make the following changes to the displayed logical properties:
   a) In the **Composition** drop-down list, click **message**.
   b) In the **Content validation** drop-down list, click **Open**, **Closed** or **Open Defined**, depending on your requirements. Note that if the embedded message is defined in a different message set, you must click **Open**.

      For further information about using these three options, see MRM content validation.

Results

Note: There are a number of different ways for the parser to identify an embedded message within a message bit stream. For further information on identifying a message within another message refer to the following concept topics.

Message Sets: Linking from one message definition file to another

Add an 'include', or an 'import' to the file that you want to reference.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Creating a message definition file" on page 2262
- "Message Sets: Opening an existing message definition file" on page 2261

About this task

There are two ways to link one message definition file to another: either you can add an 'include', or you can add an 'import', for the file that you want to reference.

To check whether a message definition file currently includes or imports other files:

1. Open the message definition file in the Message Definition editor.
2. In the Outline view, in the displayed hierarchy, select the `.mxsd` file.

3. In the Properties Hierarchy, expand **Imports** or **Includes** as appropriate to display a list of the other files that the currently selected file includes or imports.

Include

About this task

Use the include option if you want to link to a message definition file with the same namespace, or if you want to link to a message definition file with no target namespace from a message definition file with a target namespace (chameleon behavior). You must also add an include rather than an import if you want to link a message definition file with no target namespace to another message definition file that also has no target namespace.

Note: A message definition file can only reference objects in another message definition file if this other file has been included directly, so you might have a problem if you try to use the include option to include message definition files that are themselves included within other message definition files.

This task assumes that you have opened an existing message definition file.

To add an include to a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window > Show View> Outline**.
3. In the Outline view, click the message definition (`.mxsd`) file name.
4. Display the **Properties** tab of the Message Definition Editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy, right-click **Includes** then click **Add Include** on the pop-up menu.

   The "**Select Message Definition file to include**" window opens.
6. In the Message Sets pane, select the message definition file that you want to include. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**.

   The message definition file that you selected in step 4 is included within the message definition file that you opened before beginning this task.

Import

About this task

You use the import option if you want to link a message definition file to another message definition file in a different namespace. You cannot add an import from the same namespace. This restriction includes linking from a message definition file with no target namespace to another message definition file with no target namespace.

To add an import to a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Outline view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. If the Outline view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Outline**.
3. In the Outline view, click the message definition (`.mxsd`) file name.

4. Display the **Properties** tab of the Message Definition editor by clicking **Properties** in the lower-left corner of the editor area.
5. In the Properties Hierarchy, right-click **Imports** then click **Add Import**. The "**Select Message Definition file to import**" window opens.
6. In the Message Sets pane, select the message definition file that you want to import from the workspace. If the message definition files within your project are not visible in this pane, expand the project hierarchy by clicking **+**.
7. Click **Finish**.

   The message definition file that you selected in step 4 is imported into the schema of the message definition file that you opened before beginning this task.

Message Sets: Working with a message category file

This topic area lists the tasks that are involved when working with a message category file.

About this task

- "Message Sets: Creating a message category file" on page 2295
- "Message Sets: Opening an existing message category file" on page 2296
- "Message Sets: Adding a message to a message category" on page 2297
- "Message Sets: Deleting a message from a message category" on page 2297
- "Message Sets: Viewing or configuring message category file properties" on page 2298
- "Message Sets: Deleting a message category file" on page 2298

Message Sets: Creating a message category file

Create a message category file to add categories that you can use to group different message sets.

Before you begin

Complete the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

To create a message category file, complete the following steps.

Procedure

1. Open the **New Message Category File** wizard:
   a) Click **File** > **New** > **Other**.
   b) Expand **Integration Bus - Message Set Development**.
   c) Select **Message Category File**, then click **Next**.
2. In the first pane, select the `Category Kind` for the type of category that you are creating.
   - other. This value indicates that this message category represents a generic grouping of messages. The Category Usage field is disabled.
   - wsdl. This value indicates that this message category represents a WSDL operation. The specified category name is used as the WSDL operation name.

3. Optional: If you set `Category Kind` to wsdl, specify the WSDL operation type by selecting one of the following values for the `Category Usage` field, then click **Next**.
   - wsdl:request-response
   - wsdl:solicit-response
   - wsdl:one-way
   - wsdl:notification
4. Select a folder under the target message set for the new message category file to be saved. The message set folder view is filtered to show only resources in the active working set.
5. In the **File name** field, type a name for the new message category file, then click **Next**.
   The file is automatically given the file extension of `.category`.
6. Select all messages that you want to add to the new category.
   Use **Shift-click** to select a range of messages, and **Ctrl-click** to select or clear individual messages. You cannot complete the creation of the category file without adding one or more messages, and setting the **Role Type** and **Role Usage** values of each message correctly.
7. Click **Finish**.

Results

The message category file is created in the message set folder that you selected. The new message category file opens in the **Message Category** editor, so that you can view and edit it as required.

Message Sets: Opening an existing message category file

This describes how to open an existing message category file in the Message Category editor so that you can view or edit it.

Before you begin

To complete this task, you must have completed the following task:

- “Message Sets: Creating a message category file” on page 2295

About this task

To open an existing message category file:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message category file (with a file extension of `.category`) that you want to open, then click **Open** on the pop-up menu.
   This opens the message category file that you have selected in the Message Category editor.
3. View and edit the message category file as required.

Results

Tip: The Eclipse framework lets you open resource files with other editors. You are advised to only use the IBM App Connect Enterprise Toolkit Message Category editor to work with the message category files because this editor correctly validates changes made to the files. Other editors might not do this.

Message Sets: Adding a message to a message category

You can add a message to a message category file by using the Message Category editor.

Before you begin

You must have completed the following tasks:

- "Message Sets: Adding MRM message model objects" on page 2264 (to create at least one message)
- "Message Sets: Creating a message category file" on page 2295
- "Message Sets: Opening an existing message category file" on page 2296

About this task

To add a message to a message category file:

Procedure

1. Open the Message Category editor.
2. In the Properties Hierarchy, open the **Add Messages** window by right-clicking **Message Category**, then clicking **Add Messages**.

   The **Add Messages** window lists all the messages that are available for adding to the message category file. All messages that are in the message set but have not already been added to the category are displayed.
3. Select the message or messages that you would like to add.

   Use **Shift-click** to select a range of messages and **Ctrl-click** to select or clear individual messages.
4. Click **OK**.

   The selected message or messages are added to the message category and now appear in the Properties Hierarchy.

   Tip: Until you save the message category file, you can undo all additions that you make. To undo a change, right-click **Message Category** in the Properties Hierarchy, then click **Undo**. If you have added multiple messages, this action removes all the messages that you have added. If you want to remove a single message, right-click this message, then click **Undo**. To redo an addition after undoing it, use the **Redo** option.
5. Save and validate the additions that you have made to the message category file by clicking **File** > **Save**, or by pressing Ctrl+S.

Results

When you have saved the message category file after adding a message, you can no longer undo the addition of this message by using the **Undo** option. To remove a message after saving your changes, delete the message from the message category file.

When you have added a message to a message category file, you can configure its properties, according to your requirements, in the Message Category editor Details view.

Message Sets: Deleting a message from a message category

Delete a message from a message category file.

Before you begin

To complete this task, you must have completed the following tasks:

- "Message Sets: Creating a message category file" on page 2295
- "Message Sets: Opening an existing message category file" on page 2296
- "Message Sets: Adding a message to a message category" on page 2297

About this task

To delete a message from a message category file:

Procedure

In the Message Category editor, in the Properties Hierarchy, right-click the message that you want to delete, then click **Delete** on the pop-up menu.

Tip: The message is deleted from the message category file immediately, without a warning appearing first.

Results

Message Sets: Viewing or configuring message category file properties

This topic describes how to view or configure the properties of a message category file and associated messages using the Message Category editor.

Before you begin

To complete this task, you must have completed the following tasks:

- "Message Sets: Creating a message category file" on page 2295
- "Message Sets: Opening an existing message category file" on page 2296
- "Message Sets: Adding a message to a message category" on page 2297 (You must have added one or more messages to your message category file)

About this task

To configure the properties of a message category file:

Procedure

1. Switch to the Message Category editor in the Integration Development perspective.
2. To view or configure the properties of a message category, click **Message Category** in the Properties Hierarchy.

   From the Details section of the Message Category editor you can now view the properties of the message category and make any changes to the properties that are necessary.
3. To view or configure the properties of a message in the message category file, click the name of the message in the Properties Hierarchy.

   From the Details section of the Message Category editor you can now view the properties of the message and make any changes to the properties that are necessary.
4. If you have changed any of the properties in the message category or messages, you can save those changes by selecting **File** > **Save** from the menu.

What to do next

Note: Note that some combinations of `Message Category Usage`, `Role Type` and `Role Usage` are not valid for WSDL and will result in task list errors being generated.

Message Sets: Deleting a message category file

You can delete a message category file from your message model.

Before you begin

To complete this task, you must have completed the following task:

- “Message Sets: Creating a message category file” on page 2295

About this task

To delete a message category file:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message category file (`*.category` file extension) that you want to delete, then click **Delete**. Alternatively select the message category file in the Application Development view, then either click **Edit** > **Delete**, or press the Delete key.
3. On the **Confirm Resource Delete** window, click **Yes** to delete the message category file. Alternatively, to cancel the message category file deletion, either click **No** or press the Esc key.

Results

Tip: After you have deleted a message category file, the action cannot be undone.

Message Sets: Working with data structures

You can create a message definition file in a message set by importing from XML Schema, XML DTD, SCA import or export components, IBM supplied messages, WSDL definitions, IDL files, C header files, and COBOL copybooks. This topic area describes how to import from these data structures using the command line or the IBM App Connect Enterprise Toolkit.

About this task

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134.

Before you attempt to create a message definition from a data structure, by using the IBM App Connect Enterprise Toolkit, you are advised to read “Importing files from the file system into the IBM App Connect Enterprise Toolkit” on page 2319.

The following tasks topics relate to importing by using the IBM App Connect Enterprise Toolkit:

- “Message Sets: Importing from C” on page 2300
- “Message Sets: Importing from COBOL copybooks” on page 2302
- “Message Sets: Importing from IBM supplied messages” on page 2303
- “Message Sets: Importing SCA import or SCA export components” on page 2304
- “Message Sets: Generating an IBM Integration Bus SCA definition from a message set” on page 2316
- “Message Sets: Exporting an SCA Import or Export from an IBM Integration Bus SCA definition” on page 2305
- “Importing from WSDL” on page 2305
- “Message Sets: Importing an IDL file” on page 2309
- “Message Sets: Importing from XML DTD” on page 2310
- “Message Sets: Importing from XML Schema” on page 2311

The following task relates to importing by using the command line:

- “Message Sets: Importing from the command line” on page 2301 for C header files, COBOL Copybooks, XML DTDs and XML Schemas.

Message Sets: Importing from C

Create a message definition file from a C header file for use in the MRM and IDOC domains.

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134. For information about how to import a C header file for use by DFDL domain, see Creating a DFDL schema file by using the New Message Model wizard.

Before you begin

Complete the following tasks:

- "Enabling message set development" on page 2248
- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

Be aware of the following points:

- The wizard can import C header files with `.h`, `.c` and `.css` extensions. If your source file has a different extension you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file *does not* have an Custom Wire Format (CWF) layer only the logical information appears in the model. You can add the physical layer to the message set before or after importing a C header file, but you should add the physical layer *before* importing it to ensure that it is populated with settings from the C header file.
- You can import a C header file from the command line using **`mqsicreatemsgdefs`**.

About this task

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition file from a C header file:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard by clicking **FileNew Other** from the IBM App Connect Enterprise Toolkit menu.

   A window opens in which you can select a wizard.
3. Expand **Message Broker - Message Set Development**, select **Message Definition**, and click **Next**.

   The New Message Definition File wizard opens.
4. In the displayed list of options, click **C header file** then click **Next**.
5. Step through the remainder of the wizard completing the details as required.

Results

When you have completed importing the C header file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.c.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or errors have appeared.

Message Sets: Importing from the command line
This describes how to use the command line importer **mqsicreatemsgdefs** to import C, COBOL copybooks, XML DTD or XML Schema in order to populate a message set with message definitions.

Before you begin

Tip: A *message set* is the original container for message models used by IBM App Connect Enterprise. In IBM App Connect Enterprise, *message model schema* files contained in applications and libraries are the preferred way to model messages for most data formats. Message sets continue to be supported, and are required if you use the MRM or IDOC domains. If you need to model data formats for use in the MRM or IDOC domains, you must first enable message set development in the IBM App Connect Enterprise Toolkit. For more information, see "Enabling message set development" on page 2248 .

Before you attempt this task, you should read the following information:

- command

The command line importer allows you to create a new message set, into which the message definition files will be placed. When you create a new message set from the command line, only the logical information is created by default. However, the command line importer allows you to create a new message set based on an existing message set. The physical format information from the base message set is also created in the new message set. If you want physical format information to be created as well, you must take the following steps before you invoke the **mqsicreatemsgdefs** command:

1. Using the IBM App Connect Enterprise Toolkit, create a message set in your workspace that is to be used as a base message set.
2. To this base message set, add the physical formats that you want to be created in your new message set.

About this task

To import C, COBOL copybooks, XML DTD or XML Schema using the command line:

Procedure

1. Close the IBM App Connect Enterprise Toolkit.

   This must not be running when you use the command line importer.
2. Invoke the **mqsicreatemsgdefs** command from a command prompt specifying the message set project name, path name of the source files folder, and any other optional parameters that you require. If you want to add physical formats to the new message set that the **mqsicreatemsgdefs** command creates, specify the base message set that contains these physical formats as the `-base` parameter on the import command line.
3. When the command has completed, open `mqsicreatemsgdefs.report.txt`.

   This report is created when you invoke the **mqsicreatemsgdefs** command and by default is written to the directory from which you invoked the command. The report provides you with the following information:

   - Details of the parameters that were used when **mqsicreatemsgdefs** was invoked.
   - The message set level action.
   - The name of the file or files that have been imported.
   - Details of the import process (for example, any warnings that have been generated and message model objects that have been created).
   - The number of files imported.
4. Start the IBM App Connect Enterprise Toolkit and switch to the Integration Development perspective.

   The message definition file that was created when you invoked **mqsicreatemsgdefs** is visible in the project that you specified.

Results

If an error occurs during the import of a C, COBOL copybook, XML DTD, or XML Schema file, carefully check any errors that the importer reports. By default, all errors are written to the screen and to the log file described above. To gather additional information about the import, specify the `-v` (Verbose) command line parameter. This parameter displays more detailed information as the import proceeds.

Message Sets: Importing from COBOL copybooks

This topic describes how to create a new message definition from a COBOL data structure using the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit.

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134. For information about how to import a COBOL copybook for use by DFDL domain, see "Creating a DFDL schema file by using the New Message Model wizard" on page 2223. For an overview of how to use the COBOL importer, go to the Rational Application Developer for WebSphere Version 8.5.5 product documentation online and search for *COBOL Importer overview*.

Before you begin

Complete the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

Be aware of the following points:

- The wizard can import COBOL files with `.cbl`, `.ccp`, `.cob` and `.cpy` extensions. If your source file has a different extension, you must rename it before attempting to import it.
- If the message set to which you are adding the new message definition file *does not* have a Custom Wire Format (CWF) layer, or a Tagged/Delimited String (TDS) format layer, only the logical information appears in the model.

  You can add the physical layer to the message set before or after importing a COBOL data structure but ensure that you add the physical layer *before* you import the data structure to ensure that it is populated with settings from the COBOL copybook.
- You can import a COBOL data structure from the command line using **`mqsicreatemsgdefs`**.
- The copybook must not contain field names that are COBOL reserved keywords.
- The COBOL importer can only import files that have file names of 60 characters long or less. If you attempt to import a COBOL file with a name longer than 60 characters, the COBOL importer fails with the message `IWAA0652E: File name cannot be longer than 60 characters`.
- Linux The COBOL importer requires 32-bit versions of the Linux operating system libraries. Some of these libraries are not installed by default on the following distributions:

  Red Hat Enterprise Linux V6 64-bit
  Ubuntu 10.4 or later, 32-bit and 64-bit

  For details of which libraries to install, see https://www.ibm.com/support/docview.wss?uid=swg21512074

About this task

The following steps cover creating a new message definition file and overwriting the contents of an existing file.

To create a message definition file from a COBOL data structure:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard by clicking **File>New>Other** from the IBM App Connect Enterprise Toolkit menu.

   A window opens in which you can select a wizard.
3. Expand **Integration Bus - Message Set Development**, select **Message Definition File**, and click **Next**.

   The New Message Definition File wizard opens.
4. Click **COBOL file**, then click **Next**.
5. Step through the remainder of the wizard supplying the details as required.

   For more information, see Message Sets: New message definition file wizard: Create a new message definition file from a COBOL file.

Results

When you have completed importing the COBOL file using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.cobol.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or errors have appeared.

Message Sets: Importing from IBM supplied messages

You can create a new message definition file from an IBM supplied message.

Before you begin

You must have completed the following task:

- "Message Sets: Creating a message set" on page 2250

About this task

The following steps describe how to create a new message definition file, and how to overwrite the contents of an existing file.

To create a message definition from an IBM supplied message:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard by clicking **File** > **New** > **Message Definition File From** on the IBM App Connect Enterprise Toolkit menu.
3. In the displayed list of options, select **IBM supplied message** and click **Next**.
4. Complete the fields of the panel that is displayed by the wizard.

   See Message Sets: New message definition file wizard: IBM supplied message.

Results

When you have finished the import of the IBM supplied message:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project that contains the message definition that you have created. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.

- Review the messages shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or errors are displayed.

Message Sets: Importing SCA import or SCA export components

You can use the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit to import SCA import or SCA export components from WebSphere Integration Developer. You must import an SCA import or SCA export into the workspace to provide an Integration Bus SCA definition for use in configuring the SCA nodes.

Before you begin

You can import from SCA import or export by creating a message set and using the New Message Definition File wizard. Before you start, you must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

About this task

Note: From Version 7.5 onwards, WebSphere Integration Developer has been renamed IBM Integration Designer. Information in this topic that refers to WebSphere Integration Developer Version 7 is also applicable to IBM Integration Designer Version 7.5.

Ensure that the SCA import or SCA export components that you import from WebSphere Integration Developer use SOAP 1.1 bindings; a validation error occurs if you attempt to import SCA Import or SCA Export components that have been generated with SOAP 1.2 bindings.

The following steps are required to create a new message definition file, or to overwrite the contents of an existing file.

To create a message definition from an SCA import or SCA export:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File From wizard by clicking **File** > **New** > **Message Definition File From ...** on the IBM App Connect Enterprise Toolkit menu.
3. In the displayed list of options, select **SCA Import or Export** and click **Next**.

   Alternatively, open the wizard by right-clicking a file with extension `.zip` that was previously imported into the IBM App Connect Enterprise Toolkit and clicking **New> Message Definition File From ...** on the menu.
4. Step through the remainder of the wizard supplying the details as required.

   You must choose whether the SCA import or SCA export that you want to import is in the current workspace in the IBM App Connect Enterprise Toolkit or is outside the workspace.

   Check boxes provide options to:

   - Copy the source file (or files) into the 'importFiles' directory of the message set project. By default, this check box is cleared.
   - Add appropriate supported domains if they do not exist. Selecting this check box adds the XMLNSC domain for all binding types; the SOAP domain is also added for web service bindings.
   - Create an appropriate physical format if one does not exist. By default, this check box is selected.

   Note:

   - The panels and options that are available in the wizard are dependent on the settings that you select.
   - Some fields in the wizard might not be available perhaps because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

Message Sets: Exporting an SCA Import or Export from an IBM Integration Bus SCA definition

You can use the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit to export SCA import or SCA export components. You must export an SCA import or SCA export if the IBM Integration Bus SCA definition is to be used by IBM Integration Designer.

Before you begin

You must have completed the following task:

- "Message Sets: Generating an IBM Integration Bus SCA definition from a message set" on page 2316

About this task

The following steps are required to export an SCA Import or Export from an IBM Integration Bus SCA definition:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the folder that contains the SCA definition from which you want to export an SCA import or SCA export and select **Export** to start the Export wizard.
3. In the displayed list of options, select **Message Sets** > **SCA Import or Export from Integration Bus SCA Definition** and click **Next**. This action starts the SCA Import or Export from Integration Bus SCA Definition wizard.
4. Step through the remainder of the wizard and provide the details as required.

   Check boxes provide options to:

   - Overwrite existing files without warning. By default, this check box is cleared.
   - Apply working set filtering to artifact selections on this page. By default, this check box is selected.

   Note:

   - The panels and options that are available in the wizard are dependent on the settings that you select.
   - Some fields in the wizard might not be available. This might be because the field has a mandatory setting, or because the field has only one possible value, or because the field is not being used as a result of other settings that have been made.

Results

The wizard creates a folder on the file system that contains the individual files from the IBM Integration Bus SCA definition (.import or .export, .wsdl and .xsd files).

What to do next

In IBM Integration Designer, import the SCA import or export component and WSDL that have been exported from IBM App Connect Enterprise. Do not select the XSD files. See *Importing WSDL files* in the IBM Integration Designer product documentation.

Importing from WSDL

You can import a WSDL file into an application or library, or import a WSDL file into a message set.

About this task

The following sections describe how to import WSDL files:

- "Importing a WSDL file into an application or library" on page 2306
- "Importing a WSDL file into a message set" on page 2307

Before you begin

Ensure that you have completed the following tasks:

- Read about applications and libraries in "Resource management overview" on page 1942.
- Create an application or library by following the instructions in "Creating an application" on page 1963 or "Creating a library" on page 1964.

When you import a WSDL file into an application or library, all the WSDL files that are referenced are imported into the specified folder. If there are multiple WSDLs with the same name but with different paths, then the files are suffixed with an increasing index number. For example: If there are multiple files that are named `my.wsdl` in the context of import, then these files are renamed to `my1.wsdl`, `my2.wsdl`, `my3.wsdl`, and so on, in the specified folder.

About this task

To import a WSDL file into an application or library, complete the following steps.

Procedure

1. In the Application Development view, right-click an application or library, then click **New** > **Message Model**.

   The **New Message Model** wizard opens.
2. Select **SOAP XML**, then click **Next**.
3. Select **I already have WSDL for my data**, then click **Next**.
4. The name of the application or library that you selected in step 1 is shown. You can choose another application or library to contain your WSDL file, or you can create a new application or shared library by clicking **New**.
5. Optional: Specify a root destination folder. Click **Browse** to select an existing folder or create a new one.

   The imported WSDL and schema files are placed under the root folder with the same organization as the source files. The files are not placed in directories that match the namespace of the WSDL or schema files.
6. Optional: Specify an XML schema file. This field is completed automatically when you select a file from your workspace or your file system. By default, the XML schema file shares the name of the imported file.
7. Select the WSDL file to import from your workspace, or from an external URL, then click **Next**.
8. Select the bindings to import, then click **Finish**.

   For each binding that you import, a message root is created for every global element that is used by a part in the operations of the binding.

Results

Files are created and are shown under the selected application or library in the Application Development view.

What to do next

Configure an existing SOAP node by dragging a deployable WSDL file onto the node.

Before you begin

Ensure that you have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

The wizards for message set development have been provided for compatibility with previous versions of the product. To develop message sets in WebSphere Message Broker Version 8.0 and later, you must activate the message set development wizards by completing the following steps.

1. Click **Window** > **Preferences**.
2. Expand Integration Development and click **Message Sets**.
3. Select **Enable menus for Message Set development**, then click **OK**.

About this task

The following steps describe how to create a new message definition file, or overwrite the contents of an existing file.

Procedure

1. In the Application Development view, right-click the message set project, then click **New** > **Message Definition File From** > **WSDL file**.

   The **New Message Definition File** wizard opens.
2. Step through the wizard, completing the details as required.

   You must choose whether the WSDL file, or files, that you want to import are in the current workspace in the IBM App Connect Enterprise Toolkit or are outside the workspace.

   Select the appropriate check boxes to control the following behavior:

   - Copy the source file (or files) into a directory of the message set project. By default, this check box is cleared.
   - Add the SOAP and XMLNSC domains to your message set so that you can use the SOAP nodes. By default, this check box is selected.

   The panels and options that are available in the wizard depend on the settings that you select. Some fields in the wizard might not be available, because the field has a mandatory setting, the field has only one possible value, or the field is not being used as a result of other settings that have been made.

Results

When you have finished importing the WSDL file or files, complete the following steps.

- Check carefully for any errors in the report that is created when the file is imported. You can find this report in the `log` directory in the project that contains the message definition that you have created. The report has a `<wsdl-file-name>.wsdl.report.txt` file descriptor, where `<wsdl-file-name>` is the name of the WSDL definition that you are importing.
- Review the messages that are shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or errors are shown.

Required SOAP Envelope and SOAP encoding message definitions are automatically added to your message set during the import. If required, you can also import these definitions manually by using the New Message Definition File wizard, by selecting the new option **IBM supplied message**.

Message Sets: Accepting self-signed certificates when importing WSDL
You can import WSDLs that reference schemas on self-signed secure HTTPS servers, by adding security certificates to the Java Virtual Machine JVM.

About this task

The following procedure enables you to add certificates from the SSL server to each instance of your JVM.

If you are using Windows 7, you must enter the commands from a console that has administration privileges.

Procedure

1. Obtain the certificate from the server (it is a `.cer` file) and copy it into your filesystem. somewhere.

   This example uses `D:\mb.cer`
2. Open a command prompt and navigate to your Java runtime environment (JRE) bin directory that is located in your IBM App Connect Enterprise install directory, for example, `C:\Program Files\IBM\ACE\12.0.`*n*`.0\tools\common\jdk\jre\bin`.
3. Type in `keytool -printcert -file D:\mb.cer`
4. You obtain some output, and the important parameter to check is the `CN= value`.

   The value should be the same as the server name from which the WSDL is requesting files.
5. Input the certificate into a new keystore file.
   a) This procedure assumes that you can store your keystore file in `D:\mb.keystore`

      Note, that the alias must be the same name as the server and the name can be anything you require.

      For example, the name can be of the form `<userID>.<servername>.ibm.com` or `subdomain.integrationbus.com`

      The example within this topic uses the form `<userID>.<servername>.ibm.com`
   b) Type in:

   ```
   keytool -import -alias <userID>.<servername>.ibm.com -file
    D:\mb.cer -keystore D:\mb.keystore
   ```

   c) Import the certificate into a keystore file.

      You are either asked for a password, or you need to create a password when the system requests one. This is the password used in Step “7” on page 2308, and the example within this topic uses the word `password`.
   d) Select `Yes` to trust the certificate.
6. Add the keystore as an argument when you start IBM App Connect Enterprise.

   You must do this so that you can use the certificates you have just added to the keystore.
   a) Go back to `C:\Program Files\IBM\ACE\12.0.`*n*`.0\tools`.
7. Type in:

   ```
   mb -vmargs -Djavax.net.ssl.trustStore=d:\\mb.keystore
    -Djavax.net.ssl.trustStorePassword=password
   ```

8. Validate and import the WSDL

Results

You obtain a console output that is of the following format:

```
C:\Program Files\IBM\ACE\12.0.n.0\tools\common\jdk\jre\bin>keytool -printcert -file d:\
mb.cerOwner: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.
```

```
<servername>.ibm.com, OU=Integration, O=IBM,
ST=<anystate>, C=<anycountry>Issuer: EMAILADDRESS=
jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
OU=Integration, O=IBM, ST=<anystate>, C=<anycountry>
Serial number: e1cabb1486f2bc7f
Valid from: 9/27/10 12:33 PM until: 9/27/11 12:33 PM
Certificate fingerprints:
         MD5:  ED:9B:BD:1C:C7:B5:8D:6E:F3:21:B7:92:26:25:52:9B
         SHA1: 5C:DE:70:CF:A5:64:96:16:C3:ED:4E:2C:A2:6E:EA:D3:A5:4B:69:BC

C:\Program Files\IBM\ACE\12.0.n.0\tools\common\jdk\jre\bin>keytool -import -alias <userID>
.<servername>.ibm.com -file d:\mb.cer -keystore  d:\mb.keystore
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
 OU=Integration, O=IBM, ST=<anystate>, C=<anycountry>
Issuer: EMAILADDRESS=jdoe@xx.ibm.com, CN=<userID>.<servername>.ibm.com,
OU=Integration, O=IBM, ST=<anystate>, C=<anycountry>
Serial number: e1cabb1486f2bc7f
Valid from: 9/27/10 12:33 PM until: 9/27/11 12:33 PM
Certificate fingerprints:
         MD5:  ED:9B:BD:1C:C7:B5:8D:6E:F3:21:B7:92:26:25:52:9B
         SHA1: 5C:DE:70:CF:A5:64:96:16:C3:ED:4E:2C:A2:6E:EA:D3:A5:4B:69:BC
Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\Program Files\IBM\ACE\12.0.n.0\tools\common\jdk\jre\bin>cd ..

C:\Program Files\IBM\ACE\12.0.n.0\tools\common\jdk\jre>cd ..

C:\Program Files\IBM\ACE\12.0.n.0\tools\common\jdk>cd ..

C:\Program Files\IBM\ACE\12.0.n.0\tools>mb -vmargs -Djavax.net.ssl.trustStore=d:\\mb.keysto
re -Djavax.net.ssl.trustStorePassword=password

C:\Program Files\IBM\ACE\12.0.n.0\tools>
```

If an error occurs during the import of a WSDL definition, carefully check any errors that are reported. By default, all errors are written both to the screen and to the file that has the format `*.wsdl.report.txt`.

Message Sets: Importing an IDL file

You can use the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit to create a message definition from an IDL file.

Before you begin

Complete the following tasks:

- “Message Sets: Creating a message set” on page 2250
- “Importing files from the file system into the IBM App Connect Enterprise Toolkit” on page 2319
- Ensure that you have a valid IDL file. If you select an IDL file that is not valid, you see an error message and you cannot complete the wizard. When you import an IDL file, supported and unsupported operations are listed. You can import an IDL file that contains operations with types that are not supported by IBM App Connect Enterprise, but if you try to call an unsupported operation, you see an error message. The CORBA IDL file must contain at least one interface that has one operation. For more information about the IDL operations that are supported, see “CORBA support” on page 1155.
- For more information about how IDL types correspond to XML schema types, see “IDL data types” on page 1156.

About this task

The following steps describe how to use an IDL file to create a message definition file or overwrite the contents of an existing file.

Procedure

1. Right-click the message set, then click **New** > **Message Definition File From** > **CORBA IDL File** to open the New Message Definition File From wizard.

2. Complete the wizard by following the on-screen instructions.
   a) Select an IDL file either from the list of files in your workspace, or by using **Browse** to search outside your workspace.

      If you have imported an IDL file that contains includes, select the top-level IDL file.
   b) Ensure that the check box to add the DataObject domain to the message set is selected. By default, this check box is selected.
   c) Optional: You can provide a target namespace.
   d) By default, the message definition file name is the same as the name of the IDL file. You can change the name of the message definition file.
   e) If the IDL or message definition file exists, click **Next**. To rename or overwrite the existing files, select them.
3. Click **Finish**.
4. After you have imported the IDL file, check for errors.
   - Check for errors in the report that is created when the file is imported. You can find this report in the `log` directory of the project that contains the new message definition. The report is named `<idl-file-name>.idl.report.txt`, where `<idl-file-name>` is the name of the IDL file that you are importing.
   - Check for errors in the IBM App Connect Enterprise Toolkit task list.

Results

When you have finished importing the IDL file, the message definition opens. A read-only copy of the IDL file is stored in the `CORBA IDLs` folder.

For each IDL file, a single message definition is created. (If you have imported an IDL file that contains includes, all the elements and types in the IDL files are generated into a single message definition.) In the message definition, two messages are created for each operation in the IDL file (one message for the request, and one for the response), and a message is created for each user-defined exception. The request has a child element for each in and inout parameter; the response has a child element for each inout and out parameter, and a child element named "_return" for the return type of the operation.

The name of these elements is based on the interface name and operation name; for example, for the operation *sayHello* in the Interface *Hello*, the request element is called *Hello.sayHello*, and the response element is called *Hello.sayHelloResponse*. If the interface is contained in a module, the request and response element names are qualified with the names of the modules. For example, if the operation *sayHello* in the Interface *Hello* is contained in *ModuleB*, which in turn is contained in ModuleA, the response element would be called *ModuleA.ModuleB.Hello.sayHelloResponse*.

Another message definition is created with a message for each CORBA system exception.

What to do next

Develop a message flow, as described in “Developing a message flow with a CORBARequest node” on page 1163.

Message Sets: Importing from XML DTD

You can create a new message definition from an XML DTD by using the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit.

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see “Message modeling concepts” on page 2134. For information about how to import an XML DTD for use by the XMLNSC domain, see “Creating an XML schema file by using the New Message Model wizard” on page 2244.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

Before you begin this task, you should be aware of the following points:

- If the message set to which you are adding the new message definition file *does not* have an XML wire format (XML) layer only the logical information appears in the model. You can add the physical layer to the message set before or after importing from a XML DTD, but you should add the physical layer *before* importing it to ensure that it is populated with settings from the XML DTD.
- It is also possible to import an XML DTD from the command line using **mqsicreatemsgdefs**.
- The file extension must be `.dtd` in lowercase.

About this task

The following steps cover both creating a completely new message definition file and overwriting the contents of an existing file.

To create a message definition from an XML DTD:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard by clicking **File> New> Message Definition File** from the IBM App Connect Enterprise Toolkit menu.
3. In the displayed list of options, click **XML DTD file** to select it then click **Next**.
4. Step through the remainder of the wizard completing the details as required.

Results

When you have completed importing the XML DTD using the wizard:

- Carefully check for any errors in the report that is created when the file is imported. You can find this report in the `log` directory within the project containing the message definition that you have attempted to create. The report has a `.dtd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or errors have appeared.

The message definition file is created from the XML DTD and is opened in the Message Definition editor so that you can check the imported information and make any required changes. While you are checking the newly created message definition file, review any messages that appear in the IBM App Connect Enterprise Toolkit task list to see whether you need to make any corrections to resolve errors or warnings relating to the new file.

Message Sets: Importing from XML Schema

You can use the New Message Definition File wizard in the IBM App Connect Enterprise Toolkit to create a new message definition from an XML Schema

Before you begin

Tip: In IBM App Connect Enterprise, *message model schema* files contained in applications, integration services, and libraries are the preferred way to model messages for most data formats. Message sets are required if you use the MRM or IDOC domains. For more information about message modeling, see "Message modeling concepts" on page 2134. For information about how to create an XML schema, see "Creating an XML schema file by using the New Message Model wizard" on page 2244.

you must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Importing files from the file system into the IBM App Connect Enterprise Toolkit" on page 2319

Before you begin this task, you should be aware of the following points:

- If the message set to which you are adding the new message definition file *does* have an XML wire format layer, but *no* namespace support, the imported schema is modified to remove namespaces. For this reason, you should enable namespace support before importing a schema.
- If the message set to which you are adding the new message definition file *does not* have an XML wire format layer, but *does* have namespace support, only the logical information appears in the model. For this reason, you should add the physical layer to the message set before importing the schema. This ensures that the message set is populated with the settings and values from the schema. The XML Schema is not modified to remove namespaces.
- If the message set to which you are adding the new message definition file *does not* have an XML wire format layer, and *does not* have namespace support, only the logical information appears in the model and the imported schema is modified to remove namespaces.
- If you are working with a message set that does not have namespace support, you must specify the preferences that apply when you import a schema into the message set. These preferences allow you to specify how the importer treats certain individual schema constructs. You can either reject the schema if any occurrences of the construct are encountered or modify occurrences of the construct. If you choose modify, the importer modifies all occurrences of the construct.
- The extension to the XML Schema file must be `.xsd` in lowercase.

About this task

The following steps create a completely new message definition file or overwrite the contents of an existing file.

To create a message definition from an XML Schema file:

Procedure

1. Switch to the Integration Development perspective.
2. Open the New Message Definition File wizard by clicking **File> New> Message Definition File** on the IBM App Connect Enterprise Toolkit menu.

   Alternatively, you can open the wizard by right-clicking an `*.xsd` file that was previously imported into the IBM App Connect Enterprise Toolkit and clicking **New> Message Definition File** on the menu.
3. In the displayed list of options, click **XML Schema file** to select it, and then click **Next**.
4. Step through the remainder of the wizard, competing the details as required.

   The processing time for importing the XML Schema varies according to the size and complexity of that schema. In a large and complex schema, it can take some time to import the file, generate the log file and display any task list warnings or errors.

Results

When you have finished importing the XML Schema using the wizard:

- Carefully check the log file for any warnings or errors in the report that is created when the file is imported. These warnings and error messages give information about whether the schema failed to import or needed to be modified to enable it to be successfully imported. You can find this report in the `log` directory structure within the project that contains the message definition that you have tried to create. The report has a `.xsd.report.txt` file extension, prefixed with the name that you specified for the new message definition file.
- Review the messages that are shown in the IBM App Connect Enterprise Toolkit task list to check whether any new warnings or error messages have appeared. Although you might have imported a

perfectly valid schema, the task list will display warnings or error messages for any errors that exist in the message definition file. The following situations are examples where messages appear:

- If the XML Schema that you are importing contains xsd:key, xsd:keyref and xsd:unique constructs, warning messages appear in the task list to tell you that these constructs are unsupported and will be ignored by the integration server. If you prefer to delete these constructs, open the message definition file in the Message Definition editor, and delete the constructs as described in "Message Sets: Deleting objects" on page 2292. Deleting the constructs also removes the warning messages from the task list. If you decide not to delete the constructs, they remain in the message model but are not be deployed to the integration server, or used for any other purpose. The warning messages remain in the task list, but you can use the message model normally.
- If the XML Schema that you are importing contains xsd:redefine constructs, error messages appear in the task list to tell you that this construct is unsupported. If you right-click the error messages and select **Quick Fix**, you can choose to convert the xsd:redefine constructs into xsd:include constructs. This also removes the error messages.
- If the XML Schema that you are importing contains xsd:attribute constructs that contain both a fixed value and a default value, error messages appear in the task list to tell you that this construct is unsupported. However, the schema is still imported and the fixed value is used, not the default value. The error messages can be ignored.
- If you are importing a collection of related XML Schema files and the Message Definition Editor cannot resolve the links between two of the imported files, messages appear in the task list to say that referenced types or other objects cannot be found.

Message Sets: Generating documentation from message sets and message flows

You can generate documentation from your message sets, message flows, subflows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files.

About this task

To generate documentation that describes your message sets, message flows, subflows, message definition files, message maps, Java files, ESQL files, and deployable WSDL files, complete the following steps.

Bidirectional text is not fully supported in generated documentation. For example, if you enter text in the IBM App Connect Enterprise Toolkit that has a right-to-left orientation, the text is displayed in the generated documentation with a left-to-right orientation.

Procedure

1. Switch to the Integration Development perspective.
2. In the pop-up menu of the Application Development view, right-click a message set project, a message set, a message flow, a subflow, a message definition file, a Java file, an ESQL file, or a deployable WSDL file, and select the action **Generate Documentation**.

   The Documentation Generation wizard opens.
3. Provide the information that is requested to describe the documentation report that you want, and click **Next** to move to the next panel of the wizard.
4. Step through the wizard, clicking **Next** to move to a new panel, and clicking **Finish** when you have described all the information that you want your report to document.

Message Sets: Generating XML Schemas

You can generate either a single XML Schema from a message definition file, or multiple XML Schemas from a message set.

About this task

To generate a single XML Schema from a message definition file, see "Message Sets: Generating an XML Schema" on page 2315.

To generate multiple XML Schemas (one from each message definition file in a message set) see "Message Sets: Generating multiple XML Schemas" on page 2314.

Message Sets: Generating multiple XML Schemas

You can generate an XML Schema for each message definition file in a message set.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Working with a message definition file" on page 2261
- "Message Sets: Working with MRM message model objects" on page 2264

Note: IBM App Connect Enterprise uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before you generate XML Schema representations of your models.

About this task

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message set folder from which you want to generate XML Schemas, and click **Generate> XML Schemas**.
3. The **Generate XML Schemas** window is displayed, and you must put into the **Zip file name** field the name of the compressed file (`*.zip` file extension) that you want to contain the generated XML Schemas.
4. Select a destination folder for this compressed file. You can choose a location either inside or outside the workspace:
   - Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Create New Folder**.

     c. Click OK
   - Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Make New Folder** and type the name of the new folder into the directory tree.

     c. Click OK

5. Optional: Choose from the list given in the **XML Wire Format** field an XML wire format that you want to use to generate the XML Schemas.

   Tip: You must have previously added one or more XML Wire Format layers to the message set if you want to use an XML physical format when you generate XML Schemas. For further information see "Message Sets: Adding an XML wire format" on page 2256.
6. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the **Generate XML Schemas** page.

   By default, this check box is selected.

   Tip: For further information on strict and lax generation of an XML Schema, see "Generation of XML schema" on page 2205.
7. Click **Finish**.

   The compressed file that contains your generated XML Schemas is created.

Message Sets: Generating an XML Schema

You can generate an XML schema from a message definition file.

Before you begin

You must have completed the following tasks:

- "Message Sets: Creating a message set" on page 2250
- "Message Sets: Working with a message definition file" on page 2261
- "Message Sets: Working with MRM message model objects" on page 2264

Note: IBM App Connect Enterprise uses XML Schema 1.0 to describe the logical structure of messages.

Tip: You should replace any deprecated constructs before generating XML Schema representations of your models.

About this task

This task topic describes how to generate an XML Schema from a message definition file:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the message definition file (`*.msxd` file extension) from which you want to generate an XML Schema, then click **Generate** > **XML Schema** on the menu.
3. The **Generate XML Schema** window is displayed, and the message definition file that you selected is highlighted. The message definition file list is filtered to only show artifacts in the active working set. If this is not the message definition file from which you want to generate an XML Schema, select the correct message definition file.
4. Optional: From the drop down list at the bottom of the **Generate XML Schema** window, select the XML Wire Format that you want to use to generate the XML Schema.

   Tip: You must have previously added one or more XML Wire Format layers to a message set if you want to use an XML physical format when you generate XML Schema. For further information see "Message Sets: Adding an XML wire format" on page 2256.
5. If you do not want strict generation of an XML Schema, clear the **Strict generation** check box at the bottom of the **Generate XML Schema** page.

   By default, this check box is selected.

   Tip: For further information on strict and lax generation of XML Schema, see "Generation of XML schema" on page 2205.
6. Click **Next** to move to the next page of the wizard.

7. Select a destination folder for the XML Schema. You can choose a location either inside or outside the workspace:
   - Click **Create in a workspace directory** and select the required destination folder from the expanded workspace directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Create New Folder**.

     c. Click OK
   - Click **Export to an external directory** and click **Browse** to expand the directory. Select a folder from the expanded directory. The contents of the folder that you select are overwritten.

     If you want to create a new folder:

     a. Click the desired location.

     b. Click **Make New Folder** and type the name of the new folder into the directory tree.

     c. Click OK
8. Click **Finish**.

   Your XML Schema is generated.
9. Use the Application Development view to locate the destination folder that you specified for the generated XML Schema.

   This folder contains a file with exactly the same name as your message definition file with the file extension `*.xsd`. This is the generated XML Schema. To view this file, right-click it, then click **Open** on the menu. This opens the schema editor.

   Tip: The **Design**, **Source** or **Graph** tabs located in lower-left corner of the schema editor provide you with different views of generated XML Schema.

Results

Message Sets: Generating an IBM Integration Bus SCA definition from a message set

IBM App Connect Enterprise creates an IBM Integration Bus SCA definition from existing message definitions.

Before you begin

You must have created a message set that contains message definitions. The wizard gives an error if the set of message definitions is empty.

About this task

To generate an IBM Integration Bus SCA definition:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the folder that contains the message set file from which you want to generate an IBM Integration Bus SCA definition, and select **Generate** > **IBM Integration Bus SCA definition**.

   This action starts the **Generate Integration Bus SCA Definition** wizard.

3. Step through the wizard providing the details as required.

   Some of the panels and options are subject to settings that you make within the wizard and might not always be shown. Also, some fields in the wizard might be grayed out. This happens when a field has a mandatory setting, or when the field is not used because of settings that have already been made in other fields.

   By default, the wizard creates the IBM Integration Bus SCA definition in the message set project.

Results

On completion of the **Generate Integration Bus SCA Definition** wizard, you have generated either a `.insca` or a `.outsca` IBM Integration Bus SCA definition that is stored in the SCA category under the message set project.

A `.insca` IBM Integration Bus SCA definition is a compressed file that contains an SCA import and all the XSDs and WSDLs that are referenced directly or indirectly by the SCA import and a `.outsca` IBM Integration Bus SCA definition is a compressed file that contains an SCA export and all the XSDs and WSDLs that are referenced directly or indirectly by the SCA export.

Specifically, the IBM Integration Bus SCA definition contains:

- A WSDL interface that contains one or more operations.
- XSDs that correspond to the messages that are used in the operations defined in the interface.
- If you have selected a web services binding, the WSDL contains service and binding information that defines the endpoint.
- If you have not selected a web services binding, the SCA Import or Export must contain binding information for another binding that is supported by IBM App Connect Enterprise.

Message Sets: Generating a WSDL definition from a message set

To ensure the highest interoperability of your web services, use the document style of WSDL whenever possible. If rpc style WSDL is necessary, use literal encoding.

Before you begin

Before you start you must already have completed the following tasks:

- “Enabling message set development” on page 2248
- “Message Sets: Creating a message set” on page 2250

Replace any deprecated constructs before generating WSDL representations of your message models.

About this task

To generate a WSDL definition:

Procedure

1. Switch to the Integration Development perspective.
2. In the Application Development view, right-click the folder that contains the message set file from which you want to generate a web service definition, and select **Generate** > **WSDL Definition**.

   This starts the **Generate WSDL** wizard.
3. Step through the wizard completing the details as required.

   Some of the panels and options are subject to settings that you make within the wizard and might not always be shown. Also, some fields in the wizard might be greyed out. This happens when a field has

a mandatory setting, or when the field is not used because of settings that have already been made in other fields.

By default, the wizard creates the WSDL in the message set project. If you are going to use the WSDL to configure a SOAP node, create the WSDL in the message set, not the message set project.

Results

On completion of the **Generate WSDL** wizard, you have generated a WSDL definition. The file extension for WSDL files is .wsdl, and the file extension for any imported schema files in multi-file mode (where the WSDL definition is split over a number of files) is .xsd.

This following is an example of the WSDL that is generated for a JMS binding:

```
<wsdl:service name='HTTP'>
  <wsdl:port binding='tns:JMSSoapBinding' name='HTTP'>
    <wsdlsoap:address
       location='jms:/queue?destination=jms/MyQueue&amp;
                 connectionFactory=jms/MyCF&amp;
                 priority=5&amp;
                 targetService=GetQuote'/>
  </wsdl:port>
</wsdl:service>
```

Note: The various parts of the location string are broken over separate lines for clarity, but are generated as a continuous string without additional white space.

Applying a Quick Fix to a task list error

During the creation, migration and manipulation of message models, warnings or errors might occur; these are listed in the Problems view of the Integration Development perspective. Some of these warnings or errors can be cleared by applying a Quick Fix.

About this task

The types of warnings or errors that can be cleared using a Quick Fix are those where the construct has a broken link, where the construct has a facet that is not legal, or where the construct has been imported, and where a warning or error has occurred, but has been kept to ensure the integrity of structure that is being imported. This allows you to fix the problem in the most appropriate way.

To apply a Quick Fix:

Procedure

1. Switch to the Integration Development perspective.
2. Ensure that the Problems view is visible in the Integration Development perspective of the IBM App Connect Enterprise Toolkit.

   If the Problems view is not visible, from the IBM App Connect Enterprise Toolkit menu, click **Window** > **Show View** > **Problems**.
3. In the Problems view, right-click the task list warning or error that you want to apply the Quick Fix to, then click **Quick Fix**.

   Note that **Quick Fix** might not be available for the problem that you are trying to fix.
4. Step through the windows that are displayed, making the selections that are required to ensure that the fix is applied in the appropriate way.

Results

When the Quick Fix has successfully been applied to the task list warning or error, it is removed from the Problems view.

Importing files from the file system into the IBM App Connect Enterprise Toolkit

You can import files from the file system into the IBM App Connect Enterprise Toolkit by using the Import wizard, by dragging, or by copying.

About this task

Use one of the following options to import files for use by your selected project:

- "Using the Import wizard" on page 2319
- "Dragging and dropping" on page 2320
- "Copying and pasting" on page 2320

You can then select the imported file in the New Message Definition File wizard to create a message definition that is based on the contents of this file.

Using the Import wizard

About this task

Use the Import wizard to import all the files, or a selection of files, from the specified source.

To import files using the Import wizard:

Procedure

1. In the Application Development view, click the project folder into which you are going to import the files.
2. Open the Import wizard by clicking **File** > **Import**.
3. On the Select page of the Import wizard, click either **File System** or **Archive file**, depending on the type of resource that you are importing.
4. Click **Next**.
5. On the File System page, in the **Directory** field, specify the import source.

   Either type the source name in the field, or click **Browse** and select the parent directory, or compressed file that contains the file or files that you want to import. Then click **OK** (directory) or **Open** (compressed file).

   Tip: Directories from which you recently imported files are shown in the list in the **Directory** field.
6. Using the left and right panes that are displayed under the **Directory** field, specify the folders or files, or both, that you want to import.

   Consider the following points when you are making your selections:

   - To import the entire contents of a folder, select the check box for this folder in the left pane. To view secondary folders within a folder, expand the folder by clicking the plus sign (**+**).
   - To import a specific file or files within a folder, use the right pane to select the individual files that you want to import. If you select a file or files in the right pane, the check box for the folder that contains these files is disabled in the left pane to indicate that only some of the files in the folder will be imported.
   - To restrict the type of files that you are importing, click **Filter Types**, then, on the **Select Types** window, select the check boxes for the file types that you want to include, and click **OK**. If you want

to include files with extensions that are not shown in the list, type these extensions in the **Other Extensions** field.

- To select all the folders and files that are shown on the File System page, click **Select All**.
- To clear all the folders and files that are currently selected on the File System page, click **Deselect All**.

The **Select the destination for imported resources** field is already set to the name of the project folder that you selected in step 2.

7. Optional: To change the destination project or folder, click **Browse** to open the **Folder Selection** window.

   Select an alternative project folder by clicking the folder, then clicking **OK**.
8. Optional: To overwrite existing resources and not have a warning displayed, select the **Overwrite existing resources without warning** check box.

   This check box applies to both compressed files and file systems.
9. File system import only: Select one of the following options, depending on the folder structure that you want to create:

   - **Create complete folder structure**
   - **Create selected folders only**
10. Click **Finish**.

Results

The files that you selected are imported and are shown in the Application Development view under the project folder that you selected.

Dragging and dropping

About this task

You can use the drag-and-drop method to import files from your file system into the IBM App Connect Enterprise Toolkit. Drag the resources that you are importing to the exact location in the Application Development view where you want the resources to be. Do not drag them onto a blank area in the Application Development view.

To import files by dragging:

Procedure

1. In your file system, locate the file or folder that you want to import into the IBM App Connect Enterprise Toolkit.
2. Drag the file or folder to a specific location in the Application Development view.

   When you are dragging resources into the Application Development view, the project or folder into which you are trying to drop the resource is selected.
3. Ensure that the file or folder is copied into the IBM App Connect Enterprise Toolkit.

Results

Copying and pasting

About this task

You can use the copy and paste function of your operating system as a method of importing a file system into the IBM App Connect Enterprise Toolkit.

To import files by copying and pasting:

Procedure

1. Locate the file or directory that you want to import into the IBM App Connect Enterprise Toolkit.
2. Using the copy and paste function in the operating system, copy the file or directory to your system clipboard.
3. Select the destination for the file or directory in the Application Development view.
4. From the IBM App Connect Enterprise Toolkit menu, click **Edit** > **Paste**.

Results

The files or directories are copied into the IBM App Connect Enterprise Toolkit, and placed into the location that you selected.

Developing user-defined extensions

A user-defined extension is a component that you design and implement to extend the function of IBM App Connect Enterprise.

About this task

You can create and implement the following types of user-defined extension:

- Connectors
- User-defined nodes
- User-defined parsers
- User-defined exits

The user-defined extensions that you create can be used with the nodes and parsers that are supplied with the product, and with nodes and parsers that are supplied by independent software vendors.

For more information about each type of user-defined extension that you can create, see "User-defined extensions overview" on page 2321. These topics help you to understand how your user-defined extension interacts with other components of IBM App Connect Enterprise, such as message flows and their associated integration servers. A good understanding of the integration node architecture helps you to plan and construct user-defined extensions more effectively.

To create a user-defined extension, follow the instructions in the appropriate topic:

- "Developing connectors" on page 2351
- "Developing user-defined nodes from a subflow project" on page 2382
- "Developing user-defined parsers" on page 2434
- "Developing user-defined exits" on page 2443

You can write user-defined nodes in C or Java, or you can use a subflow to create a node. You write connectors in Java, and you write user-defined parsers and exits in C.

User-defined extensions overview

A user-defined extension is an optional component that you design and create to extend the functionality of IBM App Connect Enterprise.

You can create the following types of user-defined extension:

- Input nodes
- Message processing nodes
- Output nodes

- Connectors
- Parsers
- User exits

The user-defined nodes and parsers that you create can be used with the nodes and parsers that are supplied with the product, and with nodes and parsers that are supplied by other vendors. You can configure a user-defined node to use a user-defined parser.

You can write user-defined exits and parsers only in the C programming language. You can write user-defined nodes in the C or the Java programming languages, or you can use a subflow as a user-defined node. You can write connectors in the Java programming language only. You must compile user-defined nodes and parsers that are written in C into a loadable implementation library (LIL), and user exits that are written in C into a loadable exit library (LEL): that is, a shared library on Linux and UNIX systems, or a dynamic link library (DLL) on Windows systems. You must package connectors and user-defined nodes that are written in Java as a JAR file. You must import any user-defined nodes that you create into the IBM App Connect Enterprise Toolkit before you can use them. You must create a connector for the runtime environment, and a user-defined node to represent the connector in a message flow.

To achieve platform independence, use the ANSI standard C or Java programming languages, and avoid platform-specific code in your user-defined extension.

The following topics help you to understand how your user-defined extensions interact with other components of IBM App Connect Enterprise, such as message flows and integration servers. A good understanding of the integration node architecture helps you to plan and construct user-defined extensions more effectively.

- "Why use a user-defined extension?" on page 2322
- "Why use a user exit?" on page 2328
- "Which type of user-defined extension to use" on page 2331
- "Which language to use to implement a user-defined extension" on page 2350

Why use a user-defined extension?

Use a user-defined node or parser when the built-in resources do not provide the required functions.

Before you start to create your user-defined extension, be clear about its purpose. Most tasks can be performed by using the functions already provided with IBM App Connect Enterprise, but you might need a user-defined extension for your particular task.

To write user-defined extensions you need to be a skilled programmer, with some knowledge of IBM App Connect Enterprise and its architecture, therefore make sure that you have the skills and knowledge required. You also need the time to test and debug your user-defined node or parser, and a safe environment in which to do this.

Also note that the maintenance and servicing of your own user-defined extensions is your responsibility. You should ensure that there will be someone available who can perform future updates or fixes.

A user-defined extension might be appropriate in the following situations:

- When you cannot manipulate the supplied nodes or parsers to perform the function you require. For example, you might want to connect to another software component in your message flow outside of IBM MQ. If there is no supplied node for doing this, you must create your own.
- When you can improve performance, ease of use, or reliability by using your own user-defined extensions in place of the supplied nodes or parsers.
- If the available choices are not appropriate for your requirement. You can create user-defined extensions to handle internal, customer-specific, or generic commercial message formats.

Consider the following design factors when you are planning or writing a user-defined node or parser. You should be familiar with the concepts covered in the following topics before designing a user-defined extension.

Errors and exception handling

Correct handling of errors and exceptions is important for correct integration node operation. You must consider how and when your user-defined extension must handle errors and exceptions.

The errors and exception handling here describes factors that you must consider when you develop user-defined extensions for IBM App Connect Enterprise in the C programming language. If you are developing user-defined extensions using the Java programming language, you can use standard Java error and exception handling methods. If, for example, IBM App Connect Enterprise throws an exception internally, a Java exception of class **MbException** is made available.

The integration node generates C++ exceptions to handle error conditions. These exceptions are caught in the relevant software layers in the integration node, and are handled accordingly. However, programs written in C cannot catch C++ exceptions, and all exceptions thrown, by default, bypass all C user-defined extension code and are caught in a higher layer of the integration node.

Utility functions, by convention, typically use the return value to pass back requested data; for example, the address or handle of an integration node object. The return value sometimes indicates that a failure has occurred. For example, if the address or handle of an integration node object could not be retrieved, zero (CCI_NULL_ADDR) is returned. Additionally, the reason for an error condition is stored in the return code output parameter, which is, by convention, part of the function prototype of all utility functions. If the utility function completed successfully and `returnCode` was not null, `returnCode` contains CCI_SUCCESS. Otherwise, it contains one of the return codes described here. You can test the value of `returnCode` to determine whether a utility function was successful.

If the call to a utility function causes the integration node to generate an exception, the error is visible to the user-defined extension only if it specified a value for the `returnCode` parameter to that utility function. If a null value was specified for `returnCode`, and an exception occurs:

- The user-defined extension is not be aware of that exception
- The utility function does not return to the user-defined extension
- Execution control passes to higher layers in the integration node stack to process the exception

Therefore, a user-defined extension cannot perform its own error recovery. If, however, the `returnCode` parameter is specified, and an exception occurs, a return code of CCI_EXCEPTION is returned. In this case, `cciGetLastExceptionData` or `cciGetLastExceptionDataW` (the difference being that `cciGetLastExceptionDataW` returns a CCI_EXCEPTION_WIDE_ST which can contain Unicode trace text) can be used to obtain diagnostic information on the type of exception that occurred. The data is returned in the CCI_EXCEPTION_ST or CCI_EXCEPTION_WIDE_ST structure.

If there are no resources to be released, do not set the returnCode argument in your user-defined extension. Not setting this argument allows exceptions to bypass your user-defined extensions. These exceptions can then be handled higher up the IBM App Connect Enterprise stack, by the integration node.

Message inserts can be returned in the CCI_STRING_ST members of the CCI_EXCEPTION_ST structure. The CCI_STRING_ST allows the user-defined extension to provide a buffer to receive all required inserts. The integration node copies the data into this buffer, and returns the number of bytes output and the actual length of the data. If the buffer is not large enough, no data is copied and the "dataLength" member can be used to increase the size of the buffer, if required.

The user-defined extension can set a non-null value for `returnCode` and provide its own error recovery, if required. The utility function calls return to the user-defined extension and pass their status through

`returnCode`. All exceptions that occur in a utility function must be passed back to the integration node for additional error recovery to be performed; that is, when CCI_EXCEPTION is returned in `returnCode`. You do this by calling `cciRethrowLastException`, after the user-defined extension has completed its own error processing. Calling `cciRethrowLastException` causes the C interface to re-throw the last exception so that it can be handled by other layers in the integration node. In the same way as the C **`exit`** call, **`cciRethrowLastException`** does not return in this case.

If an exception occurs and is caught by a user-defined extension, the extension must not call utility functions except `cciGetLastExceptionData`, `cciGetLastExceptionDataW`, or `cciRethrowLastException`. An attempt to call other utility functions results in unpredictable behavior that can compromise the integrity of the integration node.

If a user-defined extension encounters a serious error, `cciThrowException` or `cciThrowExceptionW` can be used to generate an exception that is processed by the integration node in the correct manner. The generation of such an exception causes the supplied information to be written to the system log (syslog or Eventviewer) if the exception is not handled. The information is also written to trace (if trace is active).

Types of exception and integration node behavior

The integration node generates a set of exceptions that can be passed to a user-defined extension. These exceptions can also be generated by a user-defined extension when an error condition is encountered. The exception classes are:

Fatal
Fatal exceptions are generated when a condition occurs that prevents the integration node process from continuing execution safely, or where it is integration node policy to terminate the process. Examples of fatal exceptions are a failure to acquire a critical system resource, or an internally-caught severe software error. The integration node process terminates following the throwing of a fatal exception. Fatal exceptions are logged.

Recoverable
These exceptions are generated for errors which, although not terminal in nature, mean that the processing of the current message flow has to be ended. Examples of recoverable exceptions are invalid data in the content of a message, or a failure to write a message to an output node. When a recoverable exception is thrown, the processing of the current message is canceled on that thread, but the thread recommences execution at its input node. Recoverable exceptions are not logged.

Configuration
Configuration exceptions are generated when a configuration request fails. This can be because of an error in the format of the configuration request, or an error in the data. When a configuration exception is thrown, the request is rejected and an error response message is returned. Configuration exceptions are not logged.

Parser
These exceptions are generated by message parsers for errors that prevent the parsing of the message content or creating a bit stream. A parser exception is treated as a recoverable exception by the integration node. Parser exceptions are normally logged unless they are considered as recoverable exceptions.

Conversion
These exceptions are generated by the integration node character conversion functions if invalid data is found when trying to convert to another data type. A conversion exception is treated as a recoverable exception by the integration node and is therefore not logged.

User
These exceptions are generated when a Throw node throws a user-defined exception. User exceptions are logged.

Database
These exceptions are generated when a database management system reports an error during integration node operation. A database exception is treated as a recoverable exception by the integration node. Database exceptions are logged if they result in a fatal condition.

Storage management in user-defined nodes

Consider issues that relate to storage management when you develop user-defined extensions in the C programming language.

If you are developing user-defined extensions using the Java programming language, you can use standard Java string handling methods.

All memory that is allocated by a user-defined extension must be released by the user-defined extension. The construction of a node at run time causes the `cniCreateNodeContext` function to be invoked, which allows the user-defined extension to allocate node instance specific data areas to store a context. The address of the context is returned to the integration node, and is passed back from the integration node when an internal method causes a user-defined extension function to be invoked; thus, the C user-defined extension can locate and use the correct context for the function processing.

The integration node passes addresses of C++ objects to the user-defined extension, which are used as handles to be passed back on subsequent function calls. Your C user-defined extension must not manipulate or use these pointers in any way, for example, by trying to release storage using the `free` function. Such actions cause unpredictable behavior in the integration node.

The `cniCreateNodeContext` implementation function is invoked whenever the underlying node object has been constructed internally. It is called when an integration node is defined with a message flow that uses a user-defined node. This activity is not necessarily the same as creating (or reusing) a thread to execute a message flow instance that contains the node. The `cniCreateNodeContext` function is called only once, during the configuration of the message flow, regardless of how many threads are executing the message flow.

Similar considerations apply to user-defined parsers, and the corresponding implementation function `cpiCreateContext`.

String handling in user-defined nodes

Consider issues that relate to string handling when you develop user-defined extensions in the C programming language.

If you are developing user-defined extensions using the Java programming language, you can use standard Java string handling methods.

To enable an integration node to handle messages in all languages at the same time, text processing within the integration node is done in UCS-2 Unicode. UCS-2 Unicode character strings are also used across the Java and C language user-defined extension APIs to pass and return character data. Attributes are received in XML configuration messages as character strings, regardless of data type. If the true data type of an attribute is not a string, the `cniSetAttribute` function must perform the necessary verification and conversion before storing the attribute value. Similarly, when an attribute value is retrieved using `cniGetAttribute2`, conversion must be performed to a UCS-2 Unicode character string before returning the result.

`CciChar` defines a 16-bit character with UCS-2 Unicode representation. A `CciChar*` is a string of such characters terminated with a `CciChar` of 0. By default, a `CciChar` is represented by type `wchar_t`. However, some platforms do not have a convenient way of representing UCS-2 constants in source code, typically because of 4-byte `wchar_t` or EBCDIC representation. For example, a source-code constant such as L"ABC" expands to 12 bytes on Solaris.

For this reason, WebSphere IBM Integration provides the utility functions `cciMbsToUcs` and `cciUcsToMbs`. Use these functions, where appropriate, to ensure portability of your user-defined nodes.

Threading considerations for user-defined extensions

Message processing nodes and parsers must work in a multi-instance, multithreaded environment. Many node objects or parser objects are available, each with several syntax elements, and many threads can be executing methods on these objects.

An instance of a message flow processing node is shared and used by all the threads that service the message flow in which the node is defined. Parsers are invoked on the same thread as the nodes, therefore, if the flow is using multiple threads, the parsers are as well.

A user-defined extension must use this model. If a user-defined node requires global data or resources, you must protect the global data or resources by using semaphores to serialize access across threads. However, such serialization can result in performance bottlenecks. Avoid using global data and resources to create a more scalable solution.

The functions implemented by user-defined extensions must be reentrant, and any functions that they invoke must also be reentrant. All user-defined extension utility functions are fully reentrant.

Although a user-defined extension can create additional threads if required, all C utility functions and Java methods must be invoked on the same thread that called the `cniEvaluate` function in C or the `evaluate` method in Java, as appropriate for the language in which the node is written. If the same thread is not used, your code might compromise the integrity of the integration node and cause unpredictable behavior. Any additional threads must not call the user-defined extension API. The API must only be used from the main thread that is invoked by the Integration node.

For information about the `cniEvaluate` function see cniEvaluate.

ODBC restrictions for user-defined nodes

The ODBC environment cannot be accessed using the Java or C language user-defined extension API.

Database access must be performed using the supplied processing nodes, or by using the following implementation functions supplied for that purpose:

- cniSqlCreateStatement
- cniSqlExecute
- cniSqlSelect
- cniSqlDeleteStatement

Java Database Connectivity

Types 2 and 4 JDBC drivers are supported, but are not provided with the integration node.

User-defined extensions in the runtime environment

Before you design and implement user-defined extensions, familiarize yourself with the core components. Ensure that you also understand the basic IBM App Connect Enterprise runtime architecture.

Ensure that you are familiar with the following runtime components and concepts:

- Integration servers and integration nodes
- “User-defined extensions execution model” on page 2326

Also make sure that you understand the following concepts:

- “Message flows overview” on page 483

When you have gained an understanding of the runtime environment, read the following topics to help you understand how your user-defined extension interacts with the runtime components.

- “C user-defined input node life cycle” on page 2332
- “Java user-defined input node life cycle” on page 2333
- “C user-defined message processing nodes life cycle” on page 2336
- “Java user-defined message processing nodes life cycle” on page 2338
- “User-defined output node life cycle” on page 2343
- “User-defined parser life cycle” on page 2345

User-defined extensions execution model

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

When an integration server is initialized, the appropriate loadable implementation library (LIL) files and Plug-in Archive (PAR) files are made available to the runtime environment. The integration server runtime process starts, and creates a dedicated configuration thread. You are responsible for ensuring that a user-

defined node is thread-safe. If a node updates a variable across multiple threads, appropriate locking must be in place. Do not compromise this threading model in your implementation of user-defined nodes. Consider the following points:

- An input message sent to a message flow is processed only by the thread that received it.
- A single instance of a user-defined extension might be invoked on several threads concurrently.
- The message flow execution environment is conceptually like procedural programming. Nodes that you insert into a message flow are like subroutines called using a function call interface. However, rather than a call-return interface, in which parameters are passed in the form of input message data, the execution model is referred to as a propagation-and-return model.

As an example, consider a message flow in which you use both user-defined nodes and parsers. You use a user-defined node to process messages, and a user-defined parser to parse messages; both the node and parser contain implementation functions. The integration node calls the implementation functions, or callback functions, when certain events occur:

- When an input message is received by the message flow and is propagated to the user-defined node:
  - For C nodes, the integration node calls the `cniEvaluate` function for the user-defined node. See cniEvaluate.
  - For Java nodes, the integration node calls the `evaluate` method that is implemented by the user-defined node.
- If the user-defined node wants to query the message to decide what to do with it, the node calls a C utility function or a Java method, as appropriate for the language in which the node is written.

The integration node invokes the user-defined parser on one of its implementation functions, for example `cpiParseFirstChild`. This function instructs the parser to build the parse tree. The parser builds the tree by invoking utility functions that create elements in the parse tree, for example `cpiCreateElement`. The parser can be called many times by the integration node.

Node and parser factory behavior

The node factory and the parser factory assume roles in declaring a message flow node to the integration node or defining a parser.

Each loadable implementation library (LIL) has one node factory, or one parser factory, or has both. A node factory can identify many message flow nodes, and a parser factory can identify many parsers.

When the integration node loads the LIL, it calls the following functions:

- **`bipGetMessageflowNodeFactory`**

  After the operating system has loaded and initialized the LIL, the integration node calls initialization function `bipGetMessageflowNodeFactory`. The `bipGetMessageflowNodeFactory` function calls the utility function `cniCreateNodeFactory`, which passes back a factory name (or group name) for all the message flow nodes that your LIL supports.

- **`bipgetparserfactory`**

  After the operating system has loaded and inoitialized the LIL, the integration node calls initialization function `bipgetparserfactory`. The `bipgetparserfactory` function defines the name of the factory that the user-defined parser supports, and the classes of objects, or shared object, that the factory supports. The initialization function `bipgetparserfactory` calls the utility function `cpiCreateParserFactory`, which passes back a factory name (or group name) for all the parsers that your LIL supports.

Before the node factory is returned, the integration node calls the following functions:

1. **`cniCreateNodeFactory`**

   This function creates a single instance of the node factory in the integration node.

2. **cndDefineNodeClass**

   This function defines the name of a node class that a node factory supports, and identifies the nodes that the node factory can create.

Before the parser factory is returned, the integration node calls the following functions:

1. **cpiCreateParserFactory**

   This function creates a single instance of the named parser factory in the integration node.

2. **cpiDefineParserClass**

   This function defines the name of a parser class that a parser factory supports, and identifies the parsers that the factory can create.

See the following topics for information on these functions:

- cniCreateNodeFactory
- cpiCreateParserFactory
- cniDefineNodeClass
- cpiDefineParserClass

Why use a user exit?

Use a user exit to intercept the progress of messages through message flows without having to redesign the message flow.

User exits provide a mechanism to apply actions (such as monitoring, message tracking, and auditing) operationally to deployed message flows at run time.

You can use user exits to call (by using callbacks) your custom C code, which is provided in a loadable exit library (LEL), at key points in message transactions in deployed message flows. These user exits can use utility functions from the user-defined extensions APIs to extract details of the integration node, integration server, message flow, node, and message assembly. In addition, the user exits can use utility functions from the user-defined extensions APIs to modify parts of the message assembly.

To write user exits, you must be a skilled programmer with an understanding of IBM App Connect Enterprise and its architecture. Testing and debugging user exits can be time-consuming, and must be done in a safe environment. You must also maintain and service your own user exit.

Consider the following design factors when you plan and write a user exit:

- The effect on performance

  User exit callbacks are run inline with the current message transaction; that is, progress of the transaction is blocked until the return from the callback is received. Updating the message in a user exit callback can affect performance, particularly if the input message would not otherwise be changed in the message flow.

- Message parse timing

  On-demand parsing, referred to as partial parsing, is used to parse a message bit stream only as far as is necessary to satisfy the current reference in the message assembly. A user exit can navigate the message at each of its callback points, which can mean that the parse timing of the message flow is changed when you enable the user exit.

- Error handling

  To ensure that the error handling that is provided by the designer of a message flow that is being intercepted by a user exit continues to operate as designed, you must program the user exit in the following way:

  – All internal errors must be handled within the user exit, and the normal return from the callback must enable the message flow transaction to complete as normal.
  – All exception condition that is encountered when the user exit calls utility functions in the user-defined extensions APIs must be returned to the flow for normal error processing. This behavior is achieved by calling `cciRethrowLastException()` to cut short the callback processing.

Exploiting user exits

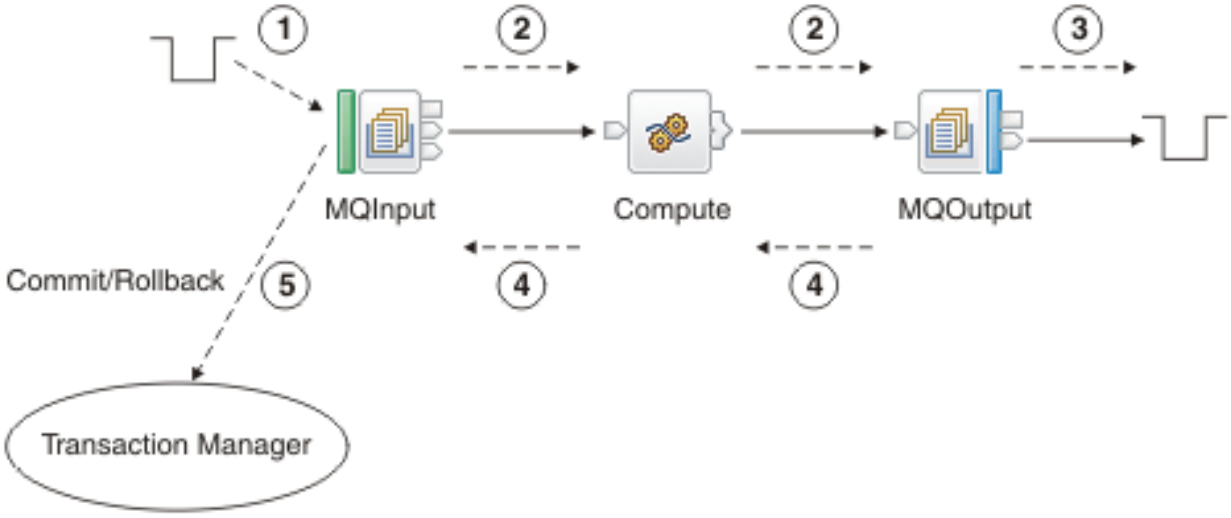
Your message flows can benefit from user exits.

Before you begin

- Read "User exits" on page 2349.
- Read "Why use a user exit?" on page 2328

About this task

The following diagram illustrates how a user exit works. The numbered events are described after the diagram. The MQInput node is used as an example, but the function applies to all input nodes, including user-defined input nodes. Similarly, the Compute and MQOutput nodes can be replaced by any equivalent nodes.



1. (`cciInputMessageCallback`) The message is dequeued from the input source (read into the flow).

   Built-in nodes and user-defined nodes differ slightly in how user exits are called. For built-in input nodes, the user exit is called as soon as possible after the data is read from the external source. For user-defined input nodes, the user exit is called just before the node propagates the message.
2. (`cciPropagatedMessageCallback`) The message is propagated to the message flow node for processing.
3. (`cciOutputMessageCallback`). A request message is sent to the output node's transport. Transport-specific destination information is written to WrittenDestination in the LocalEnvironment (for example, this information includes the queueName and msgId for an IBM MQ message). The call is made when a node successfully puts a message to a transport, from either an output or a request

node. The outputMessageEvent is called by built-in nodes only. The topic for each node that supports WrittenDestination information contains details about the data that it contains.

Note: If you use a destination list with IBM MQ or JMS nodes, a single callback is made after the message is sent to the last destination in the list. The node provides details of the message that is sent to each destination in the WrittenDestination information in the LocalEnvironment tree.

4. (`cciNodeCompletionCallback`) Node processing completes.
5. (`cciTransactionEventCallback`) The user exit is called after the transaction completes so that user exit processing is not part of that transaction. The user exit is invoked even if no transactional processing is completed by the flow.

   Where the message flow property `Commit Count` is greater than one, many-to-one ratios exist between events 1 and 5. This ratio also exists for some scenarios that are specific to the particular input node; for example, when an MQInput node is configured with the `Commit by Message Group` property selected.

You can write a user exit to track any number of these events. For each of these events, the following data is available to the user exit. All access is read-only, unless stated otherwise:

- The message is dequeued:
  - Bit stream
  - Input node
  - Environment tree (read and write)
- The message is propagated to the node:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Source node
  - Target node
- A message is sent to a transport:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Output or request node
- Message flow node processing completes:
  - Message tree (body element read and write)
  - LocalEnvironment tree (read and write)
  - Exception list
  - Environment tree (read and write)
  - Message flow node
  - Upstream node
  - Exception (if any)
- The end of the transaction:
  - Input node
  - Exception (if any)
  - Environment tree (read and write)

You can register multiple user exits, and, if they are registered, they are invoked in a defined order (see command). Any changes that are made to the message assembly (the message and environment) by a user exit are visible to subsequent user exits.

When the user exit is invoked, it can query the following information:

- Message flow information:
  - Message flow name
  - Integration node name
  - Integration node's queue manager name
  - Integration server name
  - Message flow's commit count property
  - Message flow's commit interval property
  - Message flow's coordinated transaction property
- Message flow node information:
  - Message flow node name
  - Message flow node type
  - Terminal name
  - Message flow node properties

The user exit can also do the following tasks:

- Navigate and read the message assembly (Message, LocalEnvironment, ExceptionList, Environment)
- Navigate and write the Message body, LocalEnvironment, and Environment tree.

You can register the user exits on a dynamic basis, without needing to redeploy the configuration.

Which type of user-defined extension to use

The types of user-defined extension that you can write include user-defined nodes, user-defined parsers, user-defined exits, and using a subflow as a user-defined node.

The following topics describe the different types of user-defined extension in more detail:

- "User-defined nodes" on page 2331
- "User-defined parsers" on page 2345
- "Connectors" on page 2348
- "User exits" on page 2349
- "Using a subflow as a user-defined node" on page 2344

User-defined nodes

User-defined nodes are the main mechanism for extending the functions of IBM App Connect Enterprise.

The most common uses for a user-defined node are:

- Calling an external system for which IBM App Connect Enterprise does not provide nodes
- Calling already defined program libraries that perform a transformation or calculation that is required in the design of a message flow
- Packaging a subflow

Before you consider constructing a user-defined node, make sure that no built-in node is available to perform the required actions. For example, you might have considered creating a user-defined node to perform the following tasks, but you can use a JavaCompute node instead:

- Allowing programming languages other than ESQL to be used for coding message flow functions
- Performance advantages in performing some actions in compiled code

- Complex functions that are not available in ESQL, such as the large number of classes provided in JS2E

The following topics describe the different types of user-defined node in more detail:

- "User-defined input nodes" on page 2332
- "User-defined message processing nodes" on page 2336
- "User-defined output nodes" on page 2343
- "Using a subflow as a user-defined node" on page 2344

User-defined input nodes
A user-defined input node is an extension to the integration node that provides a new input node in addition to the nodes supplied with the product.

You create user-defined input nodes by using the C or Java programming language, or from a subflow, to provide message input to a message flow from a message queue when you want your integration node to accept messages from a transport protocol other than IBM MQ.

You can use a user-defined input node to receive data from an external data source and to allow that data to be processed in an integration node. In this way, you can complement the primitive input node types provided by IBM App Connect Enterprise.

You cannot use a user-defined input node to provide the In terminal to a message subflow. If you want to provide the In terminal to a subflow, you must use the supplied Input node.

Before writing a user-defined node, make sure that you are familiar with the concepts that are introduced in "Why use a user-defined extension?" on page 2322 and "User-defined extensions in the runtime environment" on page 2326.

C user-defined input node life cycle
A user-defined input node that is written in the C programming language progresses through several stages during its lifetime.

The stages of the life cycle are:

- Registration
- Instantiation
- Processing
- Destruction

Registration

During the registration phase, the integration node discovers which resources are available and which LILs can provide them. In this instance, the resources available are nodes. The phase starts when an integration server starts. The LILs are loaded on the startup of an integration server, and the integration node queries them to find out what resources they can provide.

A CciFactory structure is created during the registration phase, when the user-defined node calls **`cniCreateNodeFactory`**.

The following APIs are called by the integration node during this stage:

- `biGetMessageflowNodeFactory`
- `bipGetParserFactory`

The following API is called by the user-defined node during this stage:

- `cniCreateNodeFactory`

Instantiation

An instance of a user-defined input node is created when the **`mqsistart`** command starts or restarts the integration server process, or when a message flow that is associated with the node is deployed.

The following APIs are called during this phase:

- **cniCreateNodeContext**. This API allocates memory for the instantiation of the user-defined node to hold the values for configured attributes. This API is called once for each message flow that is using the user-defined Input node.
- **cniCreateInputTerminal**. This API is invoked within the `cniCreateNodeContext` API, and is used to tell the integration node what input terminals, if any, your user-defined input node has.

  Your user-defined input node only has input terminals if it is also acting as a message processing node. If this is the case, it is typically better to use a separate user-defined message processing node to perform the message processing, rather than combine both operations in one, more complex, node.
- **cniCreateOutputTerminal**. This API is invoked within the `cniCreateNodeContext` API, and is used to tell the integration node what output terminals your user-defined input node has.
- **cniSetAttribute**. This API is called by the integration node to establish the values for the configured attributes of the user-defined node.

During this phase, a CciTerminal structure is created when `cniCreateTerminal` is called.

Processing

The processing phase begins when the `cniRun` function is called by the integration node. The integration node uses the `cniRun` function to determine how to process a message, including determining the domain in which a message is defined, and invoking the relevant parser for that domain.

A thread is demanded from the message flow's thread pool, and is started in the run method of the input node. The thread connects to the integration node's queue manager, and retains this connection for its lifetime. When a thread has been allocated, the node enters a message processing loop while it waits to receive a message. It remains in the loop until a message is received. If the message flow is configured to use multiple threads, thread dispatching is activated.

The message data can now be propagated downstream.

The following APIs are called by the integration node during this phase:

- **cniRun**. This function is called by the integration node to determine how to process the input message.
- **cniSetInputBuffer**. This function provides an input buffer, or tells the integration node where the input buffer is, and associates it with a message object.

Destruction

A user-defined input node is destroyed when the message flow is redeployed, or when the **mqsistop** command is used to stop the integration server process. You can destroy the node by implementing the `cniDeleteNodeContext` function.

When a user-defined input node is destroyed in one of these ways, you should free any memory used by the node, and release any held resources, such as sockets.

The following APIs are called by the integration node during this phase:

- `cniDeleteNodeContext`. This function is called by the integration node to destroy the instance of the input node.

Java user-defined input node life cycle
A user-defined input node that is written in the Java programming language progresses through several stages during its lifetime.

The stages of the life cycle are:

- Registration
- Instantiation
- Processing
- Destruction

Registration

During the registration phase a user-defined input node written in Java makes itself known to the integration node. The node is registered with the integration node through the static `getNodeName` method. Whenever an integration node starts, it loads all the relevant Java classes. The static method `getNodeName` is called at this point, and the integration node registers the input node with the node name specified in the `getNodeName` method. If you do not specify a node name, the integration node automatically creates a name for the node based on the package in which it is contained.

Using a static method here means that the method can be called by the integration node before the node itself is instantiated.

Instantiation

A Java user-defined input node is instantiated when an integration node deploys a message flow containing the user-defined input node. When the node is instantiated, the integration node calls the constructor of the input node's class.

When a node is instantiated, any terminals that you have specified are created. A message processing node can have any number of input and output terminals associated with it. You must include the `createInputTerminal` and `createOutputTerminal` methods in your node constructor to declare these terminals.

To handle exceptions that are passed back to your input node, use `createOutputTerminal` to create a catch terminal for your input node. When the input node catches an error, the catch terminal processes it in the same way as an MQInput node would. You can allow most exceptions, such as exceptions that are caused by deployment problems, to pass back to the integration node, and the integration node will warn the user of any possible configuration errors.

As a minimum, your constructor class needs only to create these output terminals on your input node. However, if you need to initialize attribute values, such as defining the parser that will initially parse a message passed from the input node, you should also include that code at this point in your input node.

Processing

Message processing for an input node begins when the integration node calls the `run` method. The `run` method creates the input message, and should contain the processing function for the input node.

The `run` method is defined in `MbInputNodeInterface`, which is the interface used in a user-defined node that defines it as an input node. You must include a `run` method in your node. If you do not include a `run` method in your user-defined input node, the node source code will not compile.

When a message flow containing a user-defined input node is deployed successfully, the integration node calls the node's run implementation method, and continues to call this method while it waits for messages to process.

When a message flow starts, a single thread is dispatched by the integration node, and is called into the input node's `run` method. If the `dispatchThread()` method is called, further threads can also be created in the same `run` method. These new threads immediately call into the input node's `run` method, and can be treated the same as the original thread. The number of new threads that can be created is defined by the `additionalInstances` property. Make sure that threads are dispatched after a message has been created, and before it is propagated, to ensure that only one thread at a time is waiting for a new message.

The user-defined input node can choose a different threading model and is responsible for implementing the chosen model. If the input node supports the `additionalInstances` property, and `dispatchThread()` is called, the code must be fully re-entrant, and any functions that are invoked by the node should also be re-entrant. If the input node forces single threading, that is, it does not call `dispatchThread()`, the node documentation must state that setting the `additionalInstances` property has no effect on the input node.

For more information on the threading model for user-defined input nodes, see "Threading considerations for user-defined extensions" on page 2325.

Destruction

A Java user-defined input node is destroyed when the node is deleted or the integration node is shut down. You do not need to include anything in your code that specifies the node should be physically deleted, because this can be handled by the garbage collector.

However, if you want notification that a node is about to be deleted, you can use the **onDelete** method. You might want to do this if there are resources that you want to delete, other than those that will be garbage collected. For example, if you have opened a socket, this will not be properly closed when the node is automatically deleted. You can include this instruction in your **onDelete** method to ensure that the socket is closed properly.

Planning user-defined input nodes
Before you develop a user-defined input node, plan and design its content and purpose.

Analysis

Before you develop a user-defined input node, ask yourself the following questions:

- Do you need to create a custom input node?

  You must include at least one input node in a message flow. Which one you choose depends on the source of the input messages:

  - If the messages arrive at the integration node on an IBM MQ queue, use the MQInput node.
  - If SOAP messages are received over HTTP, use the SOAPInput node.
  - If other messages are received over HTTP, use the HTTPInput node.
  - If the messages are received from a JMS source, use the JMSInput node.
  - If the messages are received from an EIS, use the PeopleSoftInput, SAPInput, or SiebelInput node.
  - If the messages are retrieved from files, use the FileInput node.
  - If the message source is any other, you must use a user-defined input node.

  For information about using more than one input node in a message flow, see "Using more than one input node" on page 597.
- To successfully input the data concerned, does the input node have to interface with vendor software? If so, does the API that enables access to this software break your threading model?
- Do you need a new user-defined parser to interpret the body (payload) of the message generated by this input node, or can it be parsed by a standard built-in parser?
- Do you need the user-defined input node to operate the message flow instance in which it resides under transactional control as a globally-coordinated transaction?
- Do you need the new user-defined input node to offer configuration options?
- Do you need messages propagated by this input node to be processed by the following primitives?
  - All primitive output nodes
  - ResetContentDescriptor nodes

Design considerations

Before developing and implementing your input node, decide on the following factors:

- Which message parser will initially parse the input message.
- Whether to override the default message parser attribute values for this input node.
- Which threading model is appropriate for the input node.
- What kind of message processing and transaction support will the node support.
- Which configuration attributes required by the input node should be externalized for alteration by the message flow designer.
- What optional node APIs will the user-defined node provide.

- How you will handle general development issues:
  - “Threading considerations for user-defined extensions” on page 2325
  - “Storage management in user-defined nodes” on page 2325
  - “String handling in user-defined nodes” on page 2325
  - “Errors and exception handling” on page 2323
  - Expected message formats for primitive nodes that expect specific header folders.

When you design nodes to be used as extensions to WebSphere Event Broker, the following restrictions apply:

- User-defined input nodes can support only XML, BLOB, and the IBM MQ parsers, because the MRM parser is not shipped with WebSphere Event Broker and user-defined parsers are not supported.
- User-defined nodes must not allow users to evaluate user ESQL code, because the use of ESQL in WebSphere Event Broker is not supported. For example, nodes that expose the input to **MbSQLStatement** as a node attribute are effectively emulating a Compute node.

User-defined message processing nodes
A user-defined message processing node is a node that you can create to complement the supplied built-in node types.

You might consider the use of a user-defined message processing node in the following situations:

- Your messages need transformations that the built-in nodes do not provide. For example, you might need a currency converter node.
- You want to reuse a subflow.
- You want to hide a message flow implementation by packaging it in a user-defined node.

Combine your user-defined nodes with the built-in nodes to create message flows that meet your exact business requirements.

C user-defined message processing nodes life cycle
A user-defined message processing node for the C programming language goes through various stages.

This topic covers the objects that are created and destroyed, and the implementation functions and classes that are called in the following stages:

- Registration
- Instantiation
- Processing
- Destruction

The information in this topic applies to both output nodes and message processing nodes. Both of these node types can be considered together, because although a message processing node is typically used to process a message, and an output node is used to provide an output in the form of a bit stream, you can use either type of node to perform either of these functions.

Registration

A user-defined message processing node is registered with the integration node when the LIL that contains the node has been loaded and initialized by the operating system.

The integration node calls **bipGetMessageflowNodeFactory** to establish the function of the LIL, and how the LIL should be called.

The **bipGetMessageflowNodeFactory** function in turn calls the **cniCreateNodeFactory** function, which returns a factory or group name for all of the nodes that are supported by your LIL.

The LIL should then call the utility function **cniDefineNodeClass** to pass both the name of each node and a virtual function table of the function pointers of the implementation functions.

Instantiation

During the instantiation phase, an instance of a user-defined message processing node is created. The phase starts when the integration node creates a message flow and calls the **cniCreateNodeContext** function for each instantiation of the user-defined node in that message flow. The **cniCreateNodeContext** function is that which is specified in the **iFpCreateNodeContext** field of the CNI_VFT struct passed to **cniDefineNodeClass** for that node type. This function should allocate the resources required for that node, including memory such that the instantiation of the user-defined node can hold the values for the configured attributes.

The integration node will create a node instance and call **cniCreateNodeContext** on the following occasions:

- Message flow is created:
  - Integration node is being started (user has run mqsistart). Any message flows previously deployed are re-created when the integration node starts.
  - Integration server is being reloaded (user has run mqsireload). Any message flows that have been deployed previously are re-created when the integration server reloads.
  - A severe error has occurred within the integration server which results in the integration server being restarted.
- Message flow is redeployed. When a message flow is changed and redeployed, the integration node processes redeploy by deleting all nodes in the flow and then re-creating them with the new configuration.

Note: A message flow is not created when starting an integration server. Stopping an integration server simply stops all flows and does not delete the flow or bring the process down. Restarting an integration server, starts the message flows but does not re-create the message flows.

Within **cniCreateContext**, the user-defined extension calls the two functions **cniCreateInputTerminal** and **cniCreateOutputTerminal** in order to establish what input and output terminals the message processing node has.

Processing

During the processing phase of the life cycle of a user-defined message processing node, the message is transformed in some way, when some processing operation takes place on the input message.

When the integration node retrieves a message from the queue and that message arrives at the input terminal of your user-defined node, the integration node calls the implementation function **cniEvaluate**. This function is used to decide what to do with the message.

You can use a range of node utility functions in your user-defined message processing node to perform a range of message processing functions, such as accessing the message data, accessing ESQL, transforming a message object, and propagating a message. You should include the node utility functions you are going to use to process the message within the **cniEvaluate** function.

This interface does not automatically generate a properties subtree for a message. It is not a requirement for a message to have a properties subtree, although you might find it useful to create one to provide a consistent message tree structure regardless of input node. If you want a properties subtree to be created in a message, and you are also using a user-defined input node, you must do this yourself

Destruction

When a user-defined message processing node has processed a message, you should ensure that it is destroyed, to release any system resources that it used, and to release any data areas specific to the node instance, such as context, that were acquired when the message was constructed or processed.

An instance of a user-defined message processing node is destroyed when the integration node calls the **cniDeleteNodeContext** function.

The integration node calls **cniDeleteNodeContext** when the instance of the node is deleted. The following events can cause a node to be deleted:

- Controlled termination of the integration server process:
  - Integration node is being stopped (user has run mqsistop)
  - Integration server is being reloaded (user has run mqsireload)
  - A severe error has occurred within the integration server, which results in the integration server being restarted.

  Note: This does NOT include stopping an integration server. Stopping an integration server simply stops all flows, and does not delete the flow or bring the process down.
- Message flow is deleted.
- Message flow is redeployed. When a message flow is changed and redeployed, the integration node processes redeploy by deleting all nodes in the flow and then re-creating them with the new configuration.

Java user-defined message processing nodes life cycle
During the lifecycle of the user-defined nodes that you create in Java, objects are created and destroyed, and different methods and classes called.

Every node goes through the following stages:

- Registration
- Instantiation
- Processing
- Destruction

The information here applies to both output nodes and message processing nodes. Both of these node types can be considered together, because although a message processing node is typically used to process a message, and an output node is used to provide an output, in the form of a bit stream, from a message, you can use both types of node to perform either of these functions.

Registration

The registration phase occurs when a user-defined message processing node that is written in Java contacts the integration node, or registers with the integration node.

Whenever an integration node starts, it loads all relevant LIL files and Java classes. To ensure that a message processing node is registered with the integration node, you must provide the integration node with a class that implements the `MbNodeInterface` interface and is contained in the classpath used by the integration node.

Instantiation

A Java user-defined message processing node is instantiated when an integration node deploys a message flow that contains the user-defined message processing node. When the node is instantiated, the constructor of the message processing node class is called.

When a node is instantiated, all terminals that you have specified are created. A message processing node can have an unlimited number of input and output terminals associated with it. You must include the `createInputTerminal` and `createOutputTerminal` methods in your node constructor to declare these terminals.

Output terminals include out, failure, and catch terminals. Use the `createOutputTerminal` class within the node class constructor in order to create as many output terminals as you require.

As a minimum, you must create only these output terminals by using your constructor class. However, if you need to initialize attribute values, you must also include that code at this point in your message processing node.

If you want to handle exceptions that are passed back to your message processing node, it is good practice to do this by creating a failure terminal for your user-defined message processing node, by using the `createOutputTerminal` method. It is sensible to use the failure terminal for this process because that is the terminal to which errors are propagated.

Make sure that all exceptions that are caught by the message processing node are dealt with properly. If you do not include a failure terminal, the message processing node does not attempt to handle the exception. If your message flow does not contain a method of exception handling, all exceptions thrown are passed back to the input node, where the input node deals with the exceptions.

If you do catch exceptions, make sure that you rethrow all exceptions that the message processing node cannot deal with. This action causes the exception to be passed back to the input node for handling; for example, when you want to rollback a transaction.

Processing

During the processing phase of the life cycle of a user-defined message processing node, the message processing node takes the logical hierarchy of the message and processes it in some way.

Destruction

A Java user-defined message processing node is destroyed when the node is deleted, or the integration node is shut down. You do not have to include anything in your code to specify that the node is physically deleted, because this process can be handled by the garbage collector.

However, if you want notification that a node is about to be deleted, you can use the `onDelete` method. You might want to receive notification if the node has resources that you want to delete, other than those that will be garbage collected. For example, if you have opened a socket, it is not properly closed when the node is automatically deleted. You can include this instruction in your `onDelete` method to ensure that the socket is closed properly.

Planning user-defined message processing nodes
Plan how to write your message processing node or output node, and how to navigate the message within the node.

Design factors

Before developing and implementing your message processing node, consider the following points:

- Which parser will parse messages.
- Whether to override the default message parser attribute values for this message processing node.
- What is the appropriate threading model for the message processing node.
- How to implement the end of message processing and transaction support that the node must support.
- What configuration properties required by the message processing node must be externalized for alteration by the message flow designer.
- What optional node APIs will the user-defined node provide.
- General development issues:
  - “Threading considerations for user-defined extensions” on page 2325
  - “Storage management in user-defined nodes” on page 2325
  - “String handling in user-defined nodes” on page 2325
  - “Errors and exception handling” on page 2323
  - Expected message formats for built-in nodes that expect specific header folders, see Element definitions for message parsers

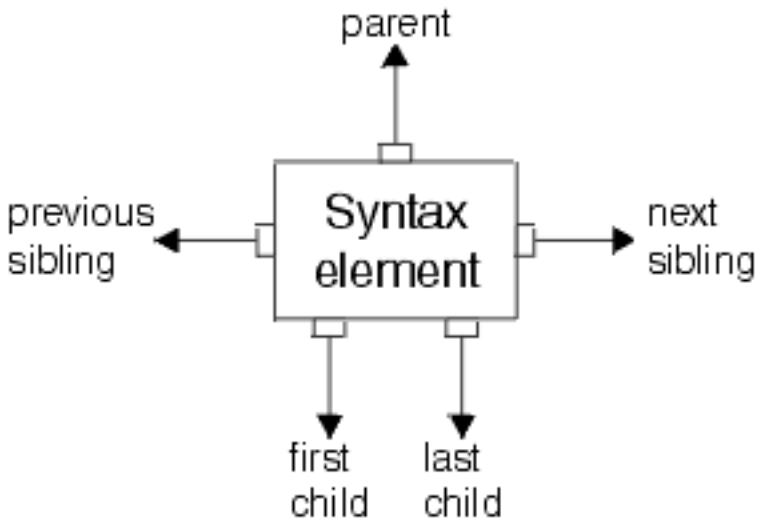
Syntax element navigation

The integration node provides functions that your node can call to traverse the tree representation of the message, as well as functions and methods that support navigation from the current element to other elements:

- Parent
- First child
- Last child

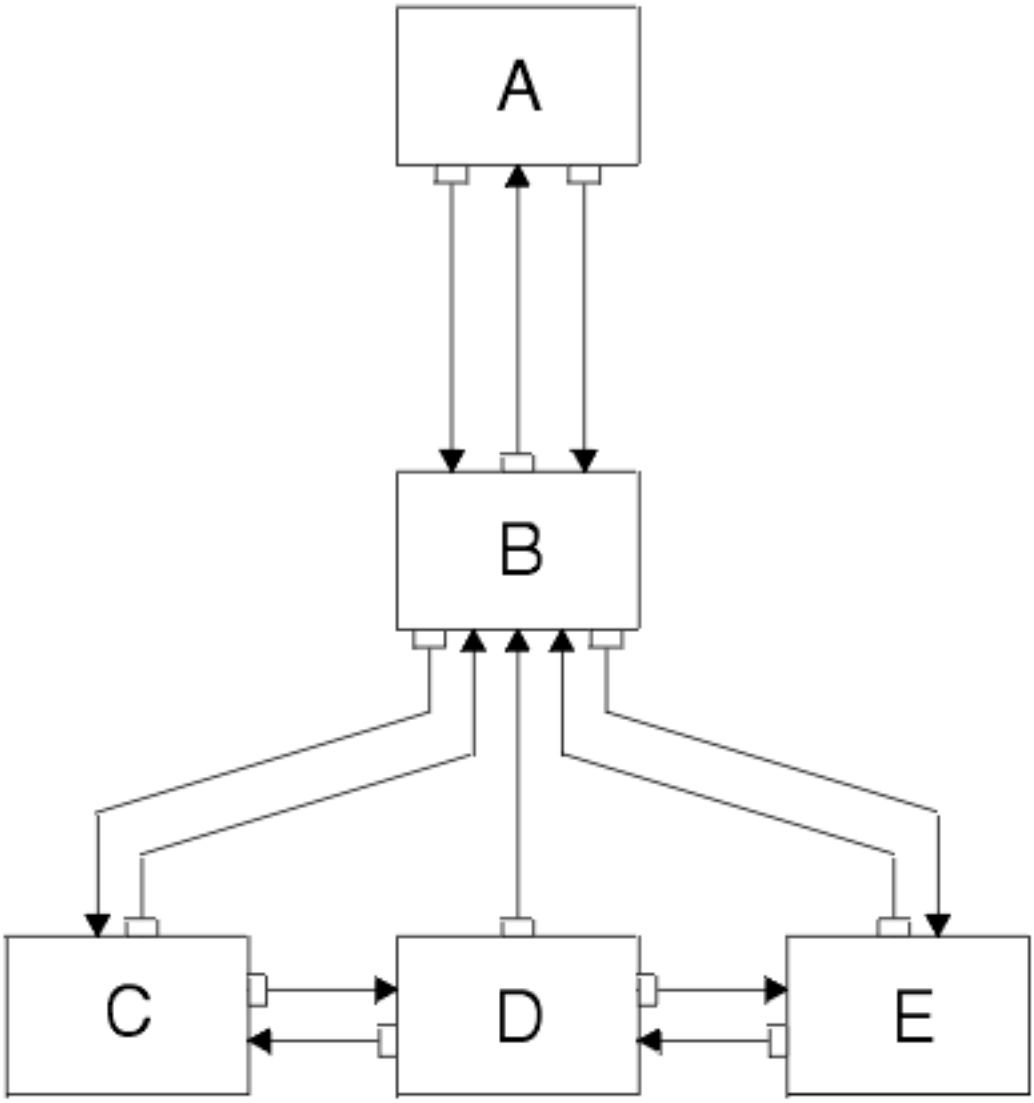
- Previous (or left) sibling
- Next (or right) sibling

These relationships are shown in the following diagram.



Other functions and methods support the manipulation of the elements themselves, with functions and methods to create elements, to set or query their values, to insert new elements into the tree, and to remove elements from the tree. See C node utility functions and C parser utility functions, or the Javadoc information for more details.

The following diagram describes a simple syntax element tree that shows a full range of interconnections between the elements.

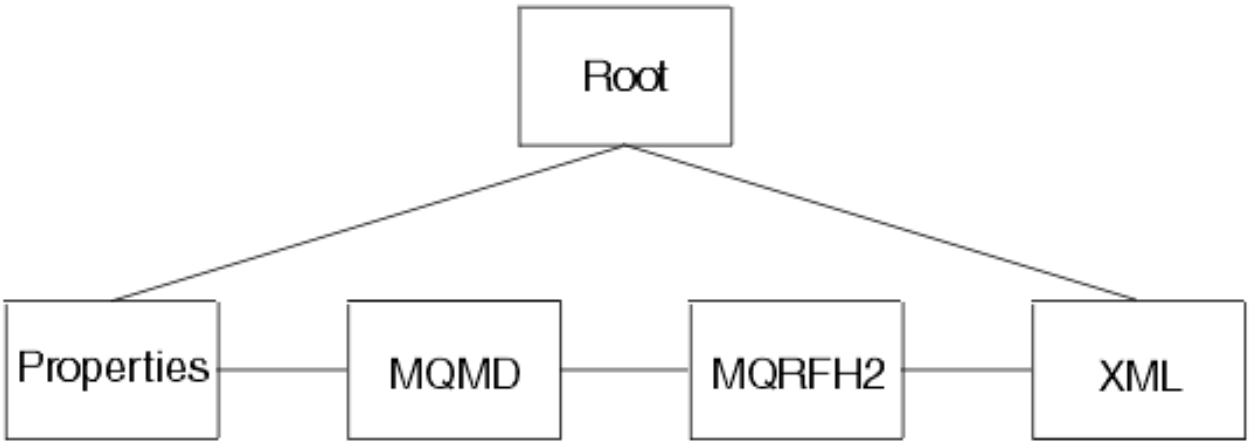


The element **A** is the root element of the tree. It has no parent because it is the root. It has a first child of element **B**. Because **A** has no other children, element **B** is also the last child of **A**.

Element **B** has three children: elements **C**, **D**, and **E**. Element **C** is the first child of **B**; element **E** is the last child of **B**.

Element **C** has two siblings: elements **D** and **E**. The next sibling of element **C** is element **D**. The next sibling of element **D** is element **E**. The previous sibling of element **E** is element **D**. The previous sibling of element **D** is element **C**.

The following diagram shows the first generation of syntax elements of a typical IBM MQ message received by an integration node. (Not all messages have an MQRFH2 header.)



These elements at the first generation are often referred to as folders, in which syntax elements that represent message headers and message content data are stored. In this example, the first child of root is the Properties folder. The next sibling of Properties is the folder for the MQMD header. The next sibling is the folder for the MQRFH2 header. The last folder represents the message content, which (in this example) is an XML message.

The previous figure includes an MQMD and an MQRFH2 header. All messages that are received by a processing node that handles IBM MQ include an MQMD header; a number of other headers can also be included.

Navigating an XML message

Consider the following XML message:

```
<Business>
  <Product type='messaging'></Product>
  <Company>
    <Title>IBM</Title>
    <Location>Hursley</Location>
    <Department>IBM MQ</Department>
  </Company>
</Business>
```

In this example, the elements are of the following types:

Name element
Business, Product, Company, Title, Location, Department

Value element
IBM, Hursley, WebSphere MQ

Name-value element
type='messaging'

Use supplied node utility functions and methods (or the similar parser utility functions) to navigate through a message. Using the XML message shown, you must call `cniRootElement` first, with the message received by the node as input to this function. In Java, you must call `getRootElement` on the incoming MbMessage object. This call returns an MbElement that represents the root of the element. Do not modify this root element in the user-defined node.

The figure of the first generation of the syntax elements of a typical message that is received by the integration node, shows that the last child of the root element is the folder containing the XML parse tree. Navigate to this folder by calling `cniLastChild` (with the output of the previous call as input to this function) in a C node, or by calling the method `getLastChild` on the root element, in a Java node.

Only one element (`<Business>`) is at the top level of the message, therefore call `cniFirstChild` (in C) or `getFirstChild` (in Java) to move to this point in the tree. Use `cniElementType` or `getType` to get its type (which is `name`), followed by `cniElementName` or `getName` to return the name itself (`Business`).

The element <Business> has two children, <Product> and <Company>. Use cniFirstChild or getFirstChild followed by cniNextSibling or getNextSibling to navigate to each child in turn.

The element <Product> has an attribute (type='messaging'), which is a child element. Use cniFirstChild or getFirstChild to navigate to this element, and cniElementType or getType to return its type (which is name-value). Use cniElementName or getName to get the name. To get the value, call cniElementValueType to return the type, followed by the appropriate function in the cniElementValue group, in this example it is cniElementCharacterValue. In Java use the method getValue, which returns a Java object representing the element value.

The element <Company> has three children, each one having a child that is a value element (IBM, Hursley, and IBM MQ). Use the functions already described to navigate to them and access their values.

Other functions are available to copy the element tree (or part of it). The copy can then be modified by adding or removing elements, and changing their names and values, to create an output message. See C node utility functions and C parser utility functions, or the Java user-defined node API, for more information.

User-defined output nodes

A user-defined output node is an extension to the integration node that provides a new message flow output node in addition to the nodes supplied with the product.

If you want your message flow to send messages by using a protocol that is not supported by IBM App Connect Enterprise, you can create your own output node.

IBM App Connect Enterprise provides the following output nodes:

- MQOutput - delivers an output message from a message flow to an IBM MQ queue
- MQReply - sends a response to the originator of the input message.
- Publication - filters output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics.
- JMSOutput - sends a message to a JMS destination
- EmailOutput - sends an email message to one or more recipients
- FileOutput - writes a message to a file

If the target application expects to receive message in any other way, you must use a user-defined output node.

User-defined output nodes can be considered together with user-defined message processing nodes. Conceptually, these two kinds of user-defined nodes are the same. Although a message processing node is typically used to process a message, and an output node is used to provide an output, in the form of a bit stream, from a message, you construct output nodes and message processing nodes in a similar way, and you can use either type of node to perform either function.

You can also create a user-defined output node from a subflow, see “Using a subflow as a user-defined node” on page 2344

For more information about user-defined output nodes, read the topics that cover user-defined message processing nodes, see “User-defined message processing nodes” on page 2336.

User-defined output node life cycle

The life cycle of a user-defined output node follows the same pattern as the life cycle of user-defined message processing nodes.

The following topics describe the life cycle of user-defined message processing nodes; read the topic that corresponds to your type of output node:

- “C user-defined message processing nodes life cycle” on page 2336
- “Java user-defined message processing nodes life cycle” on page 2338

Although a message processing node is typically used to process a message, and an output node is used to provide an output in the form of a bit stream, you can use both types of node to perform either of these functions.

Planning user-defined output nodes
A user-defined output node generates an output bit stream from a message tree.

Optionally, you can connect the node to another node and propagate the message tree for further processing. User-defined output nodes and message processing nodes are, therefore, structured in the same way. All relevant information for output nodes is included in "Planning user-defined message processing nodes" on page 2339.

Using a subflow as a user-defined node
You can develop a user-defined node that packages a subflow from scratch, in the same way that you can create any other user-defined node that has its implementation based on Java, or base it on an existing subflow.

The project that contains user-defined nodes can be exported as a plug-in that is installed in the development environment of the user. The nodes that are packaged in the plug-in are displayed in the palette in the **Message Flow** editor, and can be used in a message flow in the same way as a built-in node.

Packaging a subflow as a user-defined node provides all the benefits of a subflow, such as reusability and maintainability, as well as the following benefits:

- The user-defined node can be distributed to other developers as a plug-in.
- The user-defined node hides the implementation details of the subflow from developers who reuse the subflow.
- The user-defined node prevents developers who reuse the subflow from modifying it.
- The subflow is displayed in the palette in the Message Flow editor.

Video: Creating and using a subflow user-defined node

This video demonstrates the capability to create a subflow user-defined node using the toolkit. After creating the subflow user-defined node, it is used in another installation of App Connect Enterprise in a message flow to deploy and test the flow.

Limitations

- You cannot use user-defined nodes that are created from subflows in subflows that are defined in `.subflow` files.
- Message flows that contain user-defined nodes that are created from subflows must be included in BAR files as compiled message flow (`.cmf`) files. The shared libraries that are referenced by applications that include user-defined nodes cannot be deployed together because shared libraries need to be deployed as source. For more information, see "Adding resources to a BAR file" on page 2470.
- All the flow resources (Maps, ESQL, XSL, or other external resources), except Java code and message sets, that are referenced in the subflow, must be located in the user-defined node project.
- A user-defined node can reference another user-defined node in the same or different user-defined node project, but it must not reference anything from a regular integration project.
- The user-defined node project can have references to other projects, such as message set and Java projects.
- If the user-defined node references a message set, you must deploy the message set to the runtime separately. You can copy the message set to your workspace and deploy it through the BAR file.
- A subflow implementation of a user-defined node can contain other subflows, but all the subflows must be contained in the user-defined node project.
- Promoted properties from the nodes within the subflow are supported. Configurable promoted properties from nodes in the subflow are displayed as configurable node properties in the **BAR** editor.

- User-defined properties (UDP) on the subflow are supported. If you create multiple instances of user-defined nodes in your flow, each type of user-defined property that you define must have the same value in each instance.
- You can use the same subflow more than once to construct a flow of your own.
- You can use node types that have named correlators to create a user-defined node; for example:
  - Asynchronous request and response nodes
  - Route and Label nodes
  - Aggregation nodes
  - TimeoutControl and TimeoutNotification nodes

  These paired nodes use a unique ID string to match the pair of nodes. Therefore, you must not use more than one instance of this type of user-defined node in a flow, an integration server, or an integration node because the correlation ID will no longer be unique.
- Resources in the plug-in space are visible to all projects in the workspace. Keep user-defined nodes and their associated flows, maps, ESQL, and other similar resources, in appropriately named broker schemas. Do not put such resources in a default schema, or schemas with special names, for example, `mqsi`.

"Creating a user-defined node from a subflow" on page 2419
Create user-defined nodes either from scratch, or by using an existing subflow.

User-defined parsers

A user-defined parser is a program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A user-defined parser can also regenerate a bit stream for an outgoing message from the internal message tree representation

Create a user-defined parser when the IBM App Connect Enterprise parsers are not sufficient to parse user-defined messages.

Do not use user-defined parsers to provide connectivity or transformation functions. In most cases, the MRM or other IBM supplied parsers are capable of passing most standard type of format. You can also parse a message and construct a message tree in a user-defined node without the need to write a parser. For example, a user-defined node that reads emails from a POP3 server can parse the email and construct a message tree without the need to write a user-defined parser.

If the parser is going to be used only in a user-defined node, you do not need to use a user-defined parser. However, consider a user-defined parser if the parser will be called from other message flow nodes.

User-defined parser life cycle

Various stages exist in the life of a user-defined message flow parser.

These stages are involved:

- Registration
- Instantiation
- Processing
- Destruction

This topic describes the interactions that take place between WebSphere IBM Integration components when you run a user-defined parser. It explains each stage in terms of the events that start each stage, and the events that occur during and after each stage, and the APIs that are called. Understanding the concepts here help you to design and develop your parser more effectively.

Registration

The first phase in the user-defined parser's life cycle is the registration phase. The purpose of the registration phase is to register the user-defined parser with the integration node. This phase starts when the integration server starts.

Instantiation

The parser is created during the instantiation phase of the parser life cycle. When an input message is received, or an output message is built in a Compute node, the relevant parser is identified, and parser requirements are taken from the message header, such as the MQMD. The integration node starts and loads the Loadable Implementation Library (LIL) and the parser factory. Before the `cpiCreateContext` function is called, the integration node creates a name element as the effective root element for the parser. However, this element is not named. The parser should name this element in the `cpiSetElementName` function. The integration server process creates an instance of the parser, and the integration node makes a call to `cpiCreateContext` to allow the parser object to acquire the appropriate section of the message.

The integration node then makes a call to `cpiParseBuffer`. `cpiParseBuffer` performs any necessary initialization, and returns the length of the message content that the parser is taking ownership of. The parser assesses how much of the message data to parse, and claims the appropriate number of bytes.

Whenever an instance of a user-defined parser object is created, the context creation implementation function `cpiCreateContext` is also invoked by the integration node. This call allows the parser to allocate instance data associated with the parser. A `cpiDeleteContext` function to delete the context of the parser object is also required.

Processing

During the processing phase, the parser manipulates, alters, and references elements within the message object. The message flow processing phase begins when any message processing activity occurs, such as navigation, that requires access to an element within a message that does not exist in the integration node's internal model representation of the message concerned.

During the message flow processing phase, the parser is invoked in response to attempts to navigate into the message tree. The parser examines the buffer that was allocated when `cpiParseBuffer` was called, and creates any necessary message elements.

The parser can then navigate through the message elements, using any or all of the following parser implementation functions:

- `cpiParseFirstChild`
- `cpiParseLastChild`
- `cpiParsePreviousSibling`
- `cpiParseNextSibling`

These functions are invoked when any form of navigation is made (such as a filter expression that specifies a message field) into the part of the syntax element tree that logically represents the data for a message format supported by a user-defined parser. This navigation occurs when an operation within the integration node requires a syntax element tree to be built or extended.

Consider the following points when deciding how best to navigate the syntax element tree:

- A Syntax element has five pointers to its parents, siblings, and first and last children, so that a finite set of navigations is available.
- The same internal classes are used to perform all of these navigations.
- The parser does not control the navigation. The ESQL or a user-defined node makes the decision about the direction in which to navigate, and the order in which the navigational parser implementation functions are invoked. The user-defined parser has no control over the direction and order, and needs to respond correctly to the chosen navigation scheme; for example, parsing right to left, as well as left to right.
- When writing a user-defined parser, place the parser code in a parseNextItem function. This function should build the syntax element tree one element at a time, setting names, values and complete flags appropriately. How you implement this function depends on the nature of the bit stream to be parsed. The supplied sample parser demonstrates this behavior.

When the parser has finished parsing the relevant parts of the syntax element tree, it calls cpiWriteBuffer. This function appends its portion of the syntax element tree to the bit stream in the message buffer that is associated with the parser object, and creates the output message.

Destruction

The Destruction phase is the final phase in the user-defined parser life cycle. When the parser has written its portion of the syntax element tree to the bit stream and created the output message, the system resources that were created by the integration node for the parser to use need to be released.

The destruction phase begins when the **mqsistop** command is used to stop the execution process.

Planning user-defined parsers
Read about the concepts that you should consider before you develop a user-defined parser.

When you have considered the information provided here, and are ready to develop your own parser, use the instructions in "Developing user-defined parsers" on page 2434 to construct your parser.

Analysis

Before you start to create your own parser, be clear about its purpose. You can perform most tasks using the functions that are provided with IBM App Connect Enterprise, so you might not need to create a user-defined parser for your particular task.

Before you construct and implement a user-defined parser, consider the following questions:

- Do you need to create a user-defined parser?

  If the available parsers in IBM App Connect Enterprise are not appropriate for your needs, define your own parser to parse internal, customer-specific, or generic commercial message formats.
- Does IBM App Connect Enterprise already provide a parser for the domain or message header?

  See "Parsers" on page 520 for details of message domains for which the supplied parsers can accept input messages, and message headers with which the supplied parsers can work.
- Does the syntax of the in-house or commercial message dictate a format that can be parsed?
- To parse the message successfully, does the parser need to interact with vendor software? If so, does the API that enables access to this software break your threading model?
- Do you need to process multi-part, multi-format messages?

  IBM App Connect Enterprise does not support multi-part, multi-format messages. A multi-part MRM message must consist of messages that are all in the same format.
- What type of parsing strategy will provide best performance?

  IBM App Connect Enterprise supports partial parsing, which allows your parser to parse only relevant fields in a message. Using partial parsing can save system resources.

Partial and full parsing

IBM App Connect Enterprise supports *partial parsing*. If an individual message contains hundreds or even thousands of individual fields, the parsing operation requires considerable memory and processor resources to complete. An individual message flow might reference only a few of these fields, or none at all, so it is inefficient to parse every input message completely. For this reason, IBM App Connect Enterprise allows parsing of messages on an as-needed basis. (This ability does not prevent a parser from processing the entire message in one step, and some parsers are written to process the entire message in this way.)

Each syntax element in a logical message has two bits that indicate whether all the elements on either side of an element are complete, and whether its children are complete. Parsing is typically completed in a bottom-to-top, left-to-right manner. When a parser has parsed the siblings of a particular element that precede the given element and the first child, it sets the first completion bit to one. Similarly, when the pointer to the next sibling of an element is complete, as well as its last child pointer, the other completion bit is set to one.

In partial parsing, the integration node waits until a part of the message is referenced, and invokes the parser to parse that part of the message. Message processing nodes refer to fields within a message using hierarchical names. The name begins at the root of the message and proceeds down the message tree until the particular element is located. If an element is encountered without its completion bits set, and further navigation from this element is required, the appropriate parser entry point is called to parse the necessary part of the message. The relevant part of the message is parsed, appropriate elements are added to the logical message tree, and the element in question is marked as complete.

If you do not need to parse the full bit stream, you can use partial parsing. During partial parsing, a parser is called recursively until the requested element is returned, or until the message tree has been marked as complete, and the requested element is known not to exist.

Whether you choose to perform a full or partial parse depends on how the message will be processed. If most field elements within the message are likely to be accessed during processing, performing a full parse of the message when an attempt is made to access it is typically more efficient, particularly for smaller messages.

However, if most field elements within the message are not likely to be accessed during processing, performing a partial parse of the message when an attempt is made to access a specific field is typically more efficient, particularly when the message size grows.

Specific types used by parsers
Specific types are used when a parser needs additional information that is associated with some or all the elements in a tree in order to generate the bit stream.

For the XML parser, the specific type information is used to mark special elements such as components, processing instructions, and CDATA sections. The methods `getSpecificType` and `setSpecificType` are used by user-defined nodes to query this information and to generate message trees that use these special types.

Developers of user-defined parsers can generate their own specific type values to control special handling characteristics in their parser code using the existing C user-defined parser interface. The `getSpecificType` and `setSpecificType` methods enable Java user-defined nodes to fully use this parser capability.

Connectors
A connector is a type of user-defined extension that facilitates connection between IBM App Connect Enterprise and an external system, or *endpoint*.

Use the connector framework to develop a connector for a specific application or technology. You can use the connector framework to build reusable connectors that are not already available, reducing the need to build these interactions manually every time you need them. A connector can encapsulate complex interactions that you would otherwise achieve by using multiple message flow nodes.

You might need to use a connector in the following circumstances:

- You need to connect to an external system by using a proprietary protocol that is not supported by IBM App Connect Enterprise.
- A remote system publishes events over HTTP by using stateful, connection-oriented protocols such as XMPP or SIP. In this case, the standard IBM App Connect Enterprise HTTP nodes are not sufficient because they are stateless.
- Data from an earlier database that uses a non-standard client API or library must be integrated with later systems by using IBM App Connect Enterprise.

You create a connector for the runtime environment, and a user-defined node that represents the connector in a message flow.

For more information, see “Connectors overview” on page 2351

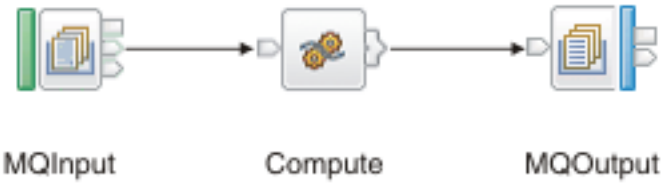
User exits

A user exit is user-provided custom software, written in C, which can be used to track data passing through message flows.

User-provided functions can be invoked at specific points during the life cycle of a message while it passes through the message flow, and can invoke utility functions to query information about the point in the flow, and the contents of the message assembly. The utility function can also modify certain parts of the message assembly. For more information about using user exits, see “Why use a user exit?” on page 2328

The user exits can be invoked when one or more of the following events occur:

- The end of a unit-of-work (UOW) or transaction (COMMIT or ROLLBACK).
- A message passes between two nodes.
- A message is successfully enqueued or sent to a transport in an output, reply, or request node.
- A message is dequeued or received in an input, response, or TimeoutNotification node.



In the basic message flow shown here, you can track messages at three levels:

- Transaction level
- Node level
- Input or output level

At the transaction level, you can track the following events:

- Messages being read into the flow
- Completion of the transaction

At the node level, you can track the following events:

- A message passing from one node to another
- Completion of processing for one node

At the message input or output level, you can track the following events:

- Messages being read into the flow
- Messages being written from the flow

Therefore, you can track five different types of event, which occur in the following sequence:

1. A message is dequeued from the input source (read into the flow).
2. A message is propagated to the node for processing.
3. A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment.
4. Node processing is completed.
5. The transaction ends.

For information about the user exit sample that is provided with IBM App Connect Enterprise, see “Transaction tracking user exit sample” on page 2446.

Which language to use to implement a user-defined extension

You can use Java or C to implement a user-defined extension.

You can use C to implement all types of user-defined extension except for connectors. You can use Java to implement user-defined nodes and connectors only.

If you can, use Java for user-defined nodes and connectors, and use C for everything else.

You must compile user-defined nodes, parsers, and exits that are written in C into a loadable implementation library (LIL): that is, a shared library on Linux and UNIX systems, or a dynamic link library (DLL) on Windows systems. You must package user-defined nodes and connectors that are written in Java as a JAR file.

To achieve platform independence, use the ANSI standard C or Java programming languages, and avoid platform-specific code in your user-defined extension.

Implementing the supplied user-defined extension samples

IBM App Connect Enterprise provides some sample code to help you understand how to write user-defined nodes and parsers.

About this task

The sample code consists of a sample parser, and the following sample nodes:

Node	Description
Switch	A node, implemented in both C and Java versions, that propagates an input message to one of several output terminals depending on the message content.
Transform	A node, implemented in both C and Java versions, that performs a simple message transformation.

Each sample node consists of the source files and some files that you can use to test each node. For the sample parser there are only source files. See Sample node files and Sample parser files for details of the sample files and where to find them.

To implement the supplied samples from the sample code that is installed with IBM App Connect Enterprise, complete the following steps:

Procedure

1. Compile the samples by following the instructions in “Compiling a Java user-defined node” on page 2418 or “Compiling a C user-defined extension” on page 2400.
2. Install the user-defined extension in an integration node domain by following the instructions in “Installing user-defined extension runtime files on an integration node” on page 2455.
3. Copy and extract the `SampleNodesProject.zip` file.

   On a computer that has the IBM App Connect Enterprise Toolkit installed, extract the `install_dir\server\sample\extensions\nodes\com.ibm.samples.nodes\SampleNodesProject.zip` file to the `dropins` directory.

   For example, on Windows, the default location to extract the file to is `C:\Program Files\IBM\ACE\12.0.n.0\tools\dropins`.
4. Restart the IBM App Connect Enterprise Toolkit, specifying the `-clean` option.

   For example, on Windows, open a command line, navigate to the installation directory (by default `C:\Program Files\IBM\ACE\12.0.n.0\tools`) and run the command **`mb.exe -clean`**. For more information about where to copy the files, see Installing a user-defined extension to current and past versions of.

5. Open the IBM App Connect Enterprise Toolkit and switch to the Integration Development perspective.

   The category called "Sample nodes" is now visible in the palette, and the sample nodes are shown below it. Documentation about the sample nodes is also visible in the help system under "Samples".
6. Include the sample nodes in a message flow.

   For more information, see “Adding a message flow node” on page 618.
7. Deploy the message flow.

   For more information, see Chapter 7, “Deploying integration solutions,” on page 2463.
8. For the Switch and Transform nodes, you can put a message to the input queue of the message flow and observe the results by completing the following steps:
   a) Ensure that the message flow that contains the sample node is deployed successfully.

      For more information, see “Checking the results of deployment” on page 2485.
   b) Use the Enqueue message function to put the sample input messages (provided in `.xml` files) to the input queue named on the input node of the message flow.

      For more information, see “Debug: putting a test message on an input queue” on page 666.

   You can also use a Trace node or the Flow debugger to see what is happening in your message flow.

Developing connectors

You can create connectors that can be used in IBM App Connect Enterprise to interact with applications or data sources.

About this task

A connector is a type of user-defined extension that facilitates connection between IBM App Connect Enterprise and an external system, or *endpoint*. A connector developer can use the connector framework to create a connector and required resources, such as a message flow node. You begin by creating a connector provider, which is the library that provides the connectors for a particular technology or type of endpoint. A connector provider has one connector factory that can create connectors of different types: input, output, and request. You then create a user-defined node to expose the capabilities of the connector to a message flow. IBM App Connect Enterprise associates a user-defined node with a runtime connector by using standard property names and values, which you define on the user-defined node.

For more information, see “Connectors overview” on page 2351.

Before you develop a connector, understand the endpoint to which you are connecting, and decide which interaction patterns are appropriate. If a connector does not exist, you can implement a connector factory, then create one or more types of connector.

To develop a connector, complete the following tasks:

- “Developing a connector for the IBM App Connect Enterprise runtime environment” on page 2353
- “Testing a connector in Eclipse (optional)” on page 2363
- “Developing a user-defined node to represent a connector” on page 2364
- “Installing a connector in IBM App Connect Enterprise” on page 2369
- “Installing the user-defined node for a connector in the IBM App Connect Enterprise Toolkit” on page 2370
- “Testing a connector in IBM App Connect Enterprise” on page 2370

Connectors overview

A *connector* helps IBM App Connect Enterprise to interact with external applications or data sources.

Companies often build their own internal applications to which they want to connect. You can connect through web services, or access data directly from a database to connect to applications. However, you can also use the connector framework to develop a connector for a specific application or technology. You can use the connector framework to build reusable connectors that are not already available, reducing

the need to build these interactions manually every time you need them. A connector can encapsulate complex interactions that you would otherwise achieve by using multiple message flow nodes.

You might need to use a connector in the following circumstances:

- You need to connect to an external system by using a proprietary protocol that is not supported by IBM App Connect Enterprise.
- A remote system publishes events over HTTP by using stateful, connection-oriented protocols such as XMPP or SIP. In this case, the standard IBM App Connect Enterprise HTTP nodes are not sufficient because they are stateless.
- Data from an earlier database that uses a non-standard client API or library must be integrated with later systems by using IBM App Connect Enterprise.

Connectors

Connectors take one of three forms, depending on the style of interaction that they use:

Input connector

Input connectors listen for external events or receive data from external sources. These connectors constitute the starting point of a message flow. Polling for input is a special case of an input connector that is supported in the connector framework.

Output connector

Output connectors send data or publish events to external systems without expecting to receive a response.

Request connector

Like output connectors, request connectors send data, but expect to receive a response in return.

Connectors for a particular application or technology are Java classes, which are packaged as a JAR file. This JAR file also contains the following resources:

- A factory class to create the connectors for that application or technology
- Supporting classes or Java libraries that are required to make the connectors function as required
- An XML descriptor file

A JAR file that provides connectors for a particular application or technology is called a *connector provider*.

Hosting connectors

Connectors constitute the runtime part of connecting to external systems or applications. You deploy connectors by installing the connector runtime files on an integration node (see “Installing a connector in IBM App Connect Enterprise” on page 2369).

During initialization, connectors get services, such as logging and security, from IBM App Connect Enterprise. A standard Java logger is provided to all connectors to allow the connector to use the logging and trace framework of IBM App Connect Enterprise.

To use connectors in IBM App Connect Enterprise, you must also create a user-defined node to expose the capabilities of the connector to a message flow.

Developing a connector for the IBM App Connect Enterprise runtime environment

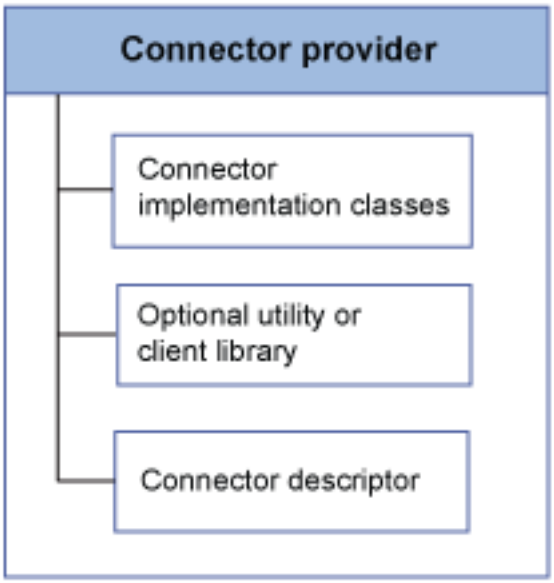
To develop a connector, you must create and compile a connector for the IBM App Connect Enterprise runtime environment, then install the connector files on an integration node.

Before you begin

For more information about connectors, see “Connectors overview” on page 2351.

About this task

To develop a connector, you create the Java classes that describe the connector factory, the connector type, and the interaction type. You also create a connector descriptor file to associate the runtime connector with a user-defined node. You can include a utility or client library. For example, you might want to use an existing library to communicate with a particular protocol, instead of creating your own.



You begin by creating the connector factory class. You then create the appropriate classes for the type of connector. A single connector provider can contain more than one connector type. You do not need to create a connector of every type for a connector provider.

Procedure

To create a connector for the runtime environment, complete the following tasks:

1. “Creating a connector factory” on page 2354.
2. Create one or more types of connector:
   - “Creating an input connector” on page 2355
   - “Creating an output connector” on page 2357
   - “Creating a request connector” on page 2359
3. “Creating a connector descriptor file” on page 2361.
4. “Packaging a connector for the runtime environment” on page 2362

Creating a connector factory

Use the connector factory to create input, output, and request connectors.

About this task

You do not need to create all types of connector for a connector provider. For example, if you want systems to send status messages to users, you might create an input connector and an output connector, which uses an instant messaging application to send messages.

Procedure

1. Create a Java project.

   For example, to create a Java project in the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Project** and select **Java Project**.

   Your Java project is shown in the Package Explorer view of the Java perspective. Right-click the project to add new classes and interfaces.

2. Add the Connector API JAR file to the class path of your Java project.

   You can find the Connector API JAR file at *install_dir*/server/classes/cmnCon.jar (for example, on Windows, C:\Program Files\IBM\IIB*version*\server\classes\cmnCon.jar).

3. Create a connector factory with the Connector Framework API by extending the AbstractConnectorFactory class.

   Javadoc documentation for the Connector Framework API is available at Connector API.

   The AbstractConnectorFactory class implements a ConnectorFactory interface. The cardinality of ConnectorFactory to connector provider is one-to-one. For example, if you want to connect to a particular resource, and you are creating a request connector and an input connector, one ConnectorFactory is required for all types of interaction.

   The following example of a connector factory Java file demonstrates how to create a connector factory that has an input, output, and request connector:

```
package connector.database;

import com.ibm.connectors.AbstractConnectorFactory;
import com.ibm.connectors.ConnectorException;
import com.ibm.connectors.InputConnector;
import com.ibm.connectors.OutputConnector;
import com.ibm.connectors.RequestConnector;

public class myConnectorFactory extends AbstractConnectorFactory {

 public myConnectorFactory() {
  // TODO You can create shared resources that are used by all connectors
  // for this provider. If the resources depend on properties that are provided
  // on initialisation, create those resources in the onInitialise() method,
  // when the properties are available.
 }

 @Override
 protected void onInitialise() throws Exception {
  // TODO This method is called after the factory has been initialised with its properties,
  // the provider name, and connector services, which are all available via getters on
  // this class. A standard Java logger is created under the provider name,
  // and it is available by using getLogger().

 }

 @Override
 public String getInfo() {
  // TODO This method returns a human-readable descriptive string for this provider,
  // such as name, version number, and organisation.
  return null;
 }

 @Override
 public InputConnector createInputConnector(String name)
   throws ConnectorException {
```

```
  // TODO This method creates and returns an input connector for this provider
  return null;
 }

 @Override
 public OutputConnector createOutputConnector(String name)
   throws ConnectorException {
  // TODO This method creates and returns an output connector for this provider
  return null;
 }

 @Override
 public RequestConnector createRequestConnector(String name)
   throws ConnectorException {
  // TODO This method creates and returns a request connector for this provider
  return null;
 }

}
```

What to do next

After you create the connector factory, create one or more connectors by following the instructions in the appropriate topic:

- “Creating an input connector” on page 2355
- “Creating an output connector” on page 2357
- “Creating a request connector” on page 2359

Creating an input connector

Create an input connector by extending the AbstractInputConnector class.

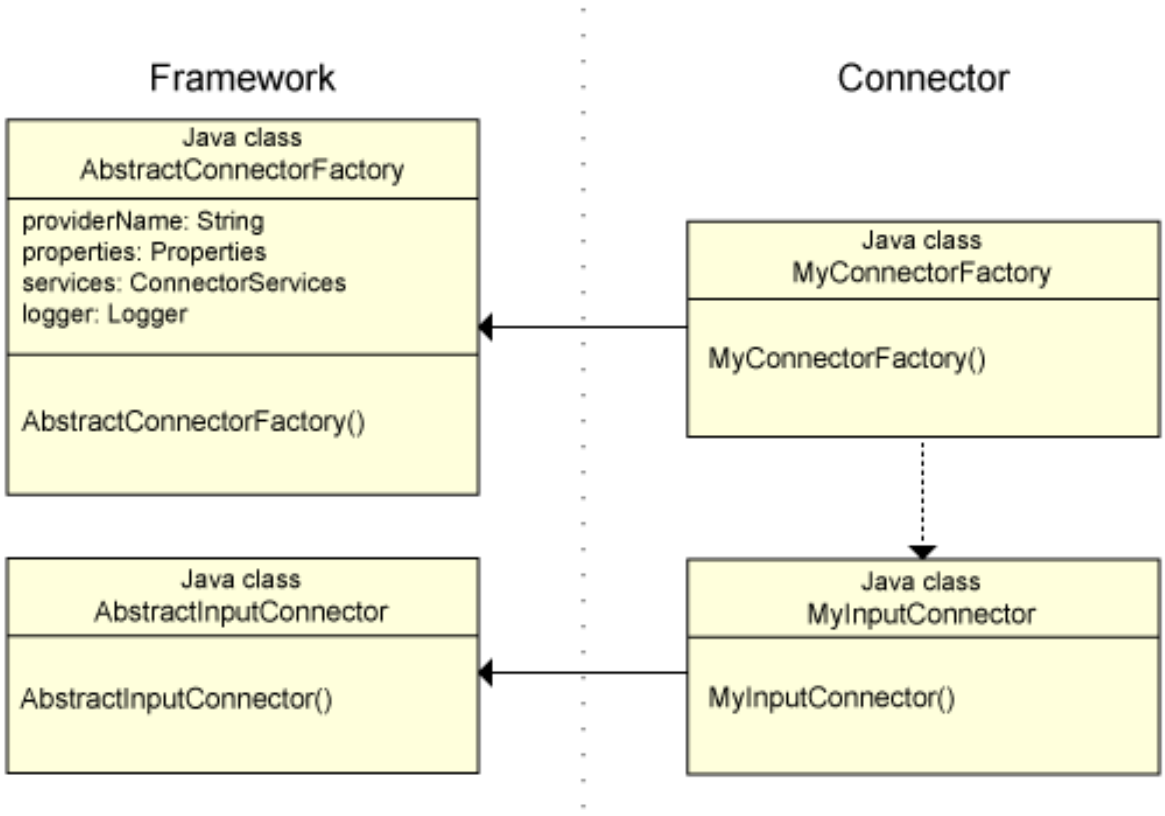
Before you begin

Create a connector provider by following the instructions in “Creating a connector factory” on page 2354.

About this task

Create an input connector when you want to listen for external events or receive data from external sources. Input connectors constitute the starting point of a message flow. Polling for input is a special case of an input connector that is supported in the connector framework.

The following example shows the Java classes that you need to create for an input connector:



Procedure

To create an input connector, extend the AbstractInputConnector class.

Javadoc documentation for the Connector Framework API is available at Connector API.

The following example of an input connector JAVA file demonstrates how to create an input connector:

```
package connector.database;

import java.util.Properties;

import com.ibm.connectors.AbstractInputConnector;
import com.ibm.connectors.ConnectorException;

public class myInputConnector extends AbstractInputConnector {

 @Override
 protected void onInitialise() throws Exception {
  // TODO This method is called after the properties and container services have been
  // saved.
  // You can call getLogger() to get a Java logger named with your connector's assigned name
  // You can call getProperty(<name>) to get a named property provided via the Toolkit
  // You can create any resources you need here (and destroy them on terminate())
  // or you can wait until start() (destroying them on stop())
  getLogger().fine("Initialisation complete");
 }

 @Override
 public void start() throws ConnectorException {
  // TODO Implement the code here to start listening for input depending on the technology
  // being used.

  // You can call getCallback().processInboundData() to pass incoming input to the host
 application
  // Input data should be a byte array. Properties may also be returned. These will form part of
 the
  // ongoing message to be processed by the message flow.
  byte[] data = null;
  Properties props = new Properties();
  getCallback().processInboundData(data, props);
```

```
 }

 @Override
 public boolean isStarted() {
  // TODO You need to track the state of the input connector since it varies depending on
  // the technology being used.
  return false;
 }

 @Override
 public void stop() throws ConnectorException {
  // TODO Close down any input listeners here.

 }

 @Override
 public void terminate() throws ConnectorException {
  // TODO Release any resources here
 }


}
```

What to do next

After you create the connector factory and connector, create the `connector.xml` descriptor file (see “Creating a connector descriptor file” on page 2361).

Creating an output connector

Create an output connector by extending the AbstractOutputConnector and AbstractOutputInteraction classes.

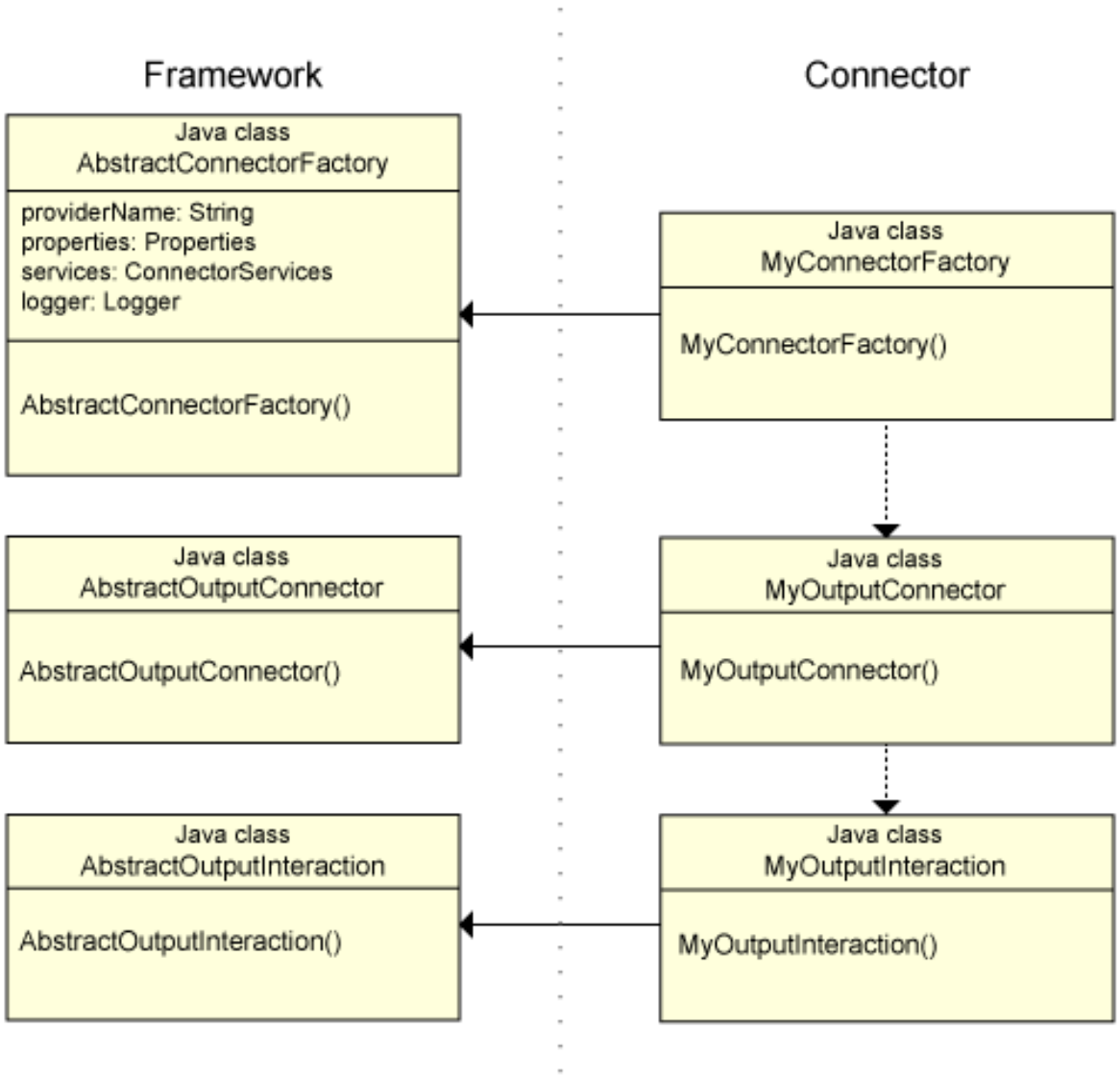
Before you begin

Create a connector provider by following the instructions in “Creating a connector factory” on page 2354.

About this task

Create an output connector when you want to send data or publish events to external systems without receiving a response.

The following example shows the Java classes that you need to create for an output connector:



Procedure

1. To create an output connector, extend the AbstractOutputConnector class.

   Javadoc documentation for the Connector Framework API is available at Connector API.

   The following example of an output connector Java file demonstrates how to create an output connector.

```
package connector.database;

import com.ibm.connectors.AbstractOutputConnector;
import com.ibm.connectors.ConnectorException;
import com.ibm.connectors.OutputInteraction;

public class myOutputConnector extends AbstractOutputConnector {

 @Override
 public OutputInteraction createOutputInteraction() throws ConnectorException {
  // TODO Create the output interaction here. The interaction is the object
  // that represents the actual output (even if it doesn't actually perform it).
  // The interaction will receive any dynamic properties configured by the message
  // flow, as well as the actual data to be sent
  return null;
 }
```

```
}
```

The connector framework calls the createOutputInteraction method when a message is to be processed.

2. Configure the output interaction by extending the AbstractOutputInteraction class.

   The following example of an output interaction Java file demonstrates how to create the output interaction:

```
package connector.database;

import java.util.Properties;

import com.ibm.connectors.AbstractOutputInteraction;
import com.ibm.connectors.ConnectorCallback;
import com.ibm.connectors.ConnectorException;

// The output interaction class publishes data to a remote system.
// Currently, the data to send is received as a byte array, and it
// should return a properties object in return that describes the outcome.
// Dynamic properties may be passed in to configure the output per-message.
public class myOutputInteraction extends AbstractOutputInteraction {

 @Override
 public Properties send(Properties overrideProperties, Object data)
   throws ConnectorException {
  byte[] byteData = (byte[]) data;
  // TODO Implement code here to send the byte data to the external
  // system. Return any details (eg success, no of bytes sent) in
  // a properties object which will be available to downstream nodes
  // within the message flow
  return null;
 }

 @Override
 public long asyncSend(Properties overrideProperties, Object data,
   ConnectorCallback callback) throws ConnectorException {
  // TODO Send asynchronously and return a 'ticket'. Currently
  // unsupported/unused.
  return 0;
 }

}
```

What to do next

After you create the connector factory and connector, create the `connector.xml` descriptor file (see “Creating a connector descriptor file” on page 2361).

Creating a request connector

Create a request connector by extending the AbstractRequestConnector and AbstractRequestInteraction classes.

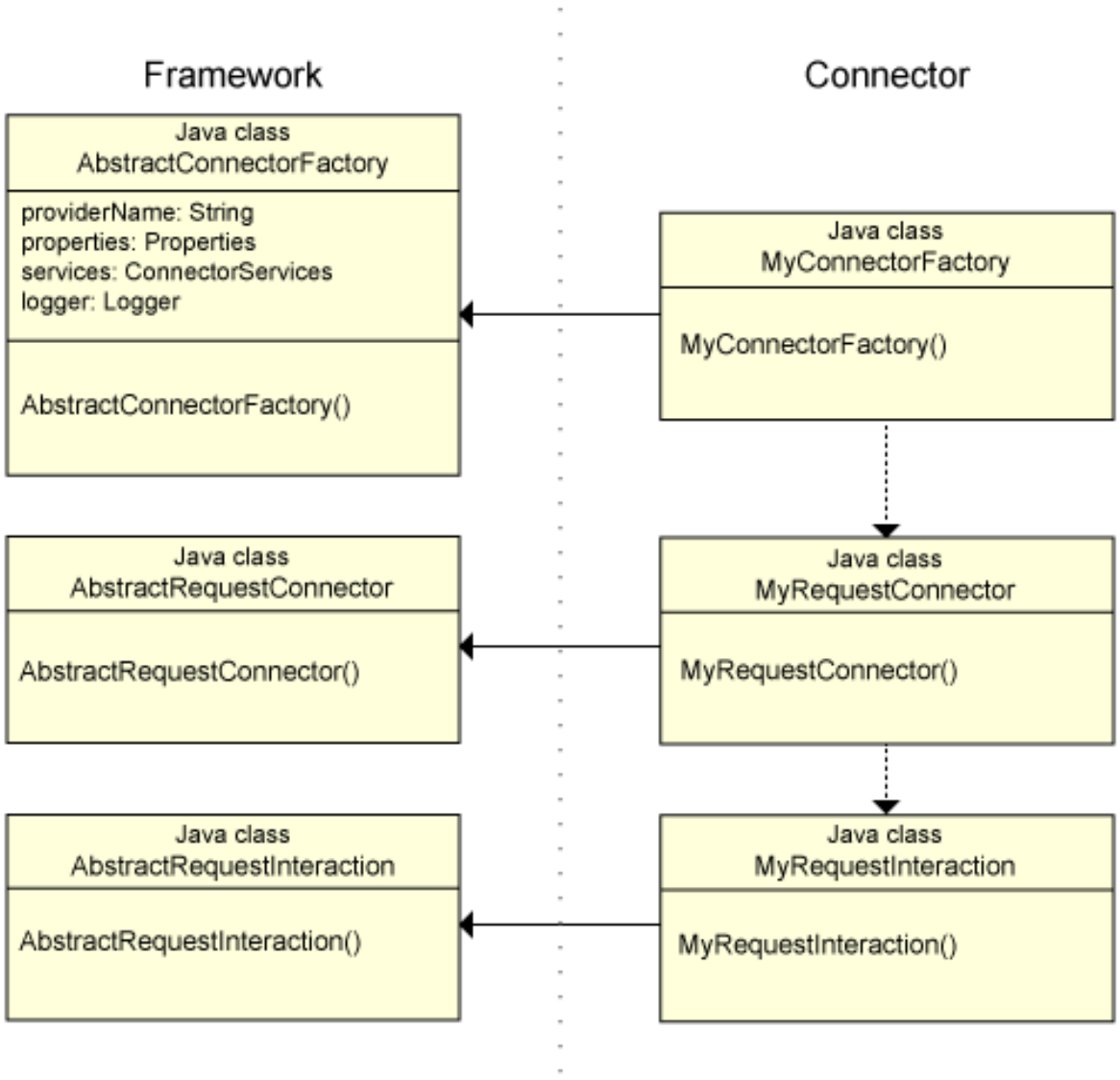
Before you begin

Create a connector factory by following the instructions in “Creating a connector factory” on page 2354.

About this task

Create a request connector when you want to send data, or publish events to external systems, and receive a response.

The following example shows the Java classes that you need to create for a request connector:



Javadoc documentation for the Connector Framework API is available at Connector API.

Procedure

To create a request connector, complete the following steps.

1. Create the connector by extending the AbstractRequestConnector class.

   The following example request connector Java file demonstrates how to create a request connector:

```
package connector.database;

import com.ibm.connectors.AbstractRequestConnector;
import com.ibm.connectors.ConnectorException;
import com.ibm.connectors.RequestInteraction;

public class myRequestConnector extends AbstractRequestConnector {

 @Override
 public RequestInteraction createRequestInteraction() throws ConnectorException {
  // TODO Create the request interaction here. The interaction is the object
  // that represents the actual request (even if it doesn't actually perform it).
  // The interaction will receive any dynamic properties configured by the message
  // flow, as well as the actual data to be sent
  return null;
 }
```

```
}
```

The connector framework calls the createRequestInteraction method when a message is to be processed.

2. Specify the request connector interaction by extending the AbstractRequestInteraction class.

   The following example of a request interaction Java file demonstrates how to create the request interaction:

```
package connector.database;

import java.util.Properties;

import com.ibm.connectors.AbstractRequestInteraction;
import com.ibm.connectors.ConnectorCallback;
import com.ibm.connectors.ConnectorException;

// The request interaction class performs a request/response with a remote system.
// Currently, data is received as a byte array, and should return a byte array in
// response. Dynamic properties may be passed in to configure the request per-message.
public class myRequestInteraction extends AbstractRequestInteraction {

 @Override
 public Object request(Properties overrideProperties, Object data)
   throws ConnectorException {
  byte[] input = (byte[]) data;
  // TODO Implement code here to send the byte data to the external
  // system. Return the response which currently must be a byte array.
  byte[] output = new byte[0];
  return output;
 }

 @Override
 public long asyncRequest(Properties overrideProperties, Object data,
   ConnectorCallback callback) throws ConnectorException {
  // TODO Request asynchronously and return a 'ticket'. Currently
  // unsupported/unused.
  return 0;
 }

}
```

What to do next

After you create the connector factory and connector, create the `connector.xml` descriptor file (see “Creating a connector descriptor file” on page 2361).

Creating a connector descriptor file

A connector descriptor file is used to associate the runtime connector with a user-defined node. This file creates a link between the logic in a message flow and the code that is called at run time.

Before you begin

Create a connector factory and appropriate connectors. For more information, see “Developing a connector for the IBM App Connect Enterprise runtime environment” on page 2353.

About this task

The `connector.xml` descriptor file contains the elements **`connectorProvider`** and **`connectorFactory`**:

- The **`connectorProvider`** element contains the name of the provider, which can include multiple interaction types. Therefore, the provider name is the same if you are creating an input connector, an output connector, and a request connector for that provider.
- The **`connectorFactory`** element identifies the fully qualified name of the provider's factory class (for example, `connector.database.myConnectorFactory`). This property identifies the implementation class that extends the AbstractConnectorFactory class in the Connector framework.
- The **`connectorFactory properties`** are available to the connector factory during initialization.

IBM App Connect Enterprise associates the runtime connector with a user-defined node by using these properties. The **connectorProvider** element in the `connector.xml` file must match the **connectorName** property on the user-defined node.

Procedure

To create the `connector.xml` descriptor file, complete the following steps.

1. Create an XML file that is called `connector.xml` and specify the **connectorProvider** and **connectorFactory** elements.

   Here is an example of a `connector.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
 <connectorProvider name="myDatabaseConnector">
     <connectorFactory className="connector.database.myConnectorFactory">
         <properties>
             <property name="name1" value="value1"/>
         </properties>
     </connectorFactory>
 </connectorProvider>
```

2. Save the `connector.xml` file in your Java project, at the root of the `src` folder.

What to do next

After you create the connector factory, connectors, and connector descriptor file, package the appropriate files (see "Packaging a connector for the runtime environment" on page 2362.

Packaging a connector for the runtime environment

Package your connector so that it can be installed in a runtime environment.

Before you begin

Create the connector factory, connectors, and connector descriptor file. For more information, see "Developing a connector for the IBM App Connect Enterprise runtime environment" on page 2353.

About this task

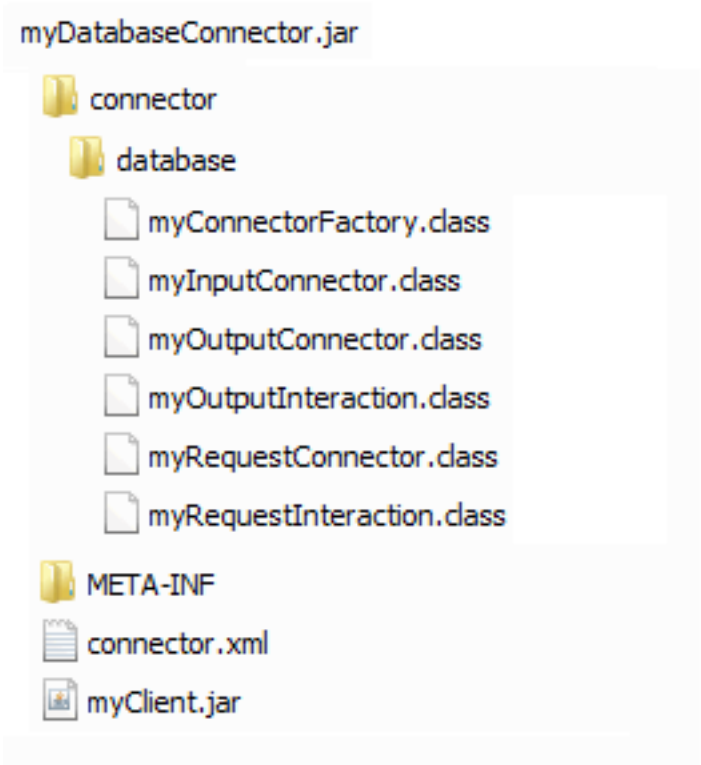
Procedure

To package the connector, complete the following steps:

1. In Eclipse, click **File** > **Export**.
2. In the **Export** wizard, expand **Java**, and select **JAR file**, then click **Next**.
3. Select your Java project from the list of files in your workspace.
4. Select **Export all output folders for checked projects**.
5. Select the location in the file system for your JAR file.

Results

Your connector is packaged into a JAR file and saved at the specified location. The following example shows the structure of files in the JAR file, and includes an input, output, and request connector. In this example, `myClient.jar` is a library that is used by the connectors. You can create your own library, or acquire one from a third party.



What to do next

After you package the connector, install it in a runtime environment. For example, see "Installing a connector in IBM App Connect Enterprise" on page 2369.

Testing a connector in Eclipse (optional)

To test a connector outside a container such as IBM App Connect Enterprise, create a JUnit test case and run that test case in Eclipse.

Before you begin

- Complete the connector development steps for the runtime environment in "Developing a connector for the IBM App Connect Enterprise runtime environment" on page 2353.
- Ensure that you have access to the endpoint with which your connector interacts.

About this task

When you create a connector, you can test it by creating a JUnit test case. You can run the test case in Eclipse, without the need to install a container, such as IBM App Connect Enterprise.

Procedure

To test a connector in Eclipse, complete the following steps:

1. Create some mock classes for your connector.
2. Create a JUnit test case to test the methods of your classes.
3. To run the test, right-click the Java test case in the Package Explorer or Application Development view and click **Run as** > **JUnit Test**.

Results

The output from the test in the console window shows a number of interactions with the endpoint.

Developing a user-defined node to represent a connector

To use a connector in a message flow, you must create, package, and install a user-defined node for the IBM App Connect Enterprise Toolkit

Before you begin

Create the connector for the runtime environment by completing the tasks in "Developing a connector for the IBM App Connect Enterprise runtime environment" on page 2353.

About this task

To use a connector in a message flow, you create a user-defined node in the IBM App Connect Enterprise Toolkit. You package the node, then distribute it so that message flow developers can install it in the appropriate location.

To implement a connector in IBM App Connect Enterprise, complete the following tasks:

Procedure

1. "Creating a user-defined node for a connector" on page 2364
2. "Configuring the user-defined node properties and terminals for your connector" on page 2365
3. "Packaging the user-defined node project for a connector" on page 2368

Creating a user-defined node for a connector

Create a user-defined node that uses a connector in a message flow.

Before you begin

Create the connector factory, connectors, and connector descriptor for the runtime environment by completing the tasks in "Developing a connector for the IBM App Connect Enterprise runtime environment" on page 2353.

About this task

The project that contains a user-defined node is a user-defined node project. When you create a user-defined node, you can add it to an existing user-defined node project, or you can create a new project. A user-defined node project can contain more than one user-defined node. Typically, all user-defined nodes for a particular connector provider are contained in the same project. For example, if you create an input, output, and request connector for a connector provider, all three types of connector have a user-defined node in the same project.

Procedure

To create a user-defined node, complete the following steps:

1. In the IBM App Connect Enterprise Toolkit, click **File** > **New** > **User-defined Node**.

   The **New User-defined Node** wizard opens.
2. Select an existing user-defined node project, or click **New** to create a new project.

   The following steps assume that you create a new project.
3. In the **New User-Defined Node Project** wizard, select **Create a new category** and enter a name for the category.

   This category is displayed on the message flow node palette.
4. Enter a project name.
   For example, use the same name as the connector provider.

   When you enter the project name and move to another field, the other fields in the wizard are filled automatically.
5. Click **Finish**.

6. In the **New User-Defined Node** wizard, specify a schema for this node.

   Do not use the default schema or any other common schema, such as `mqsi`, which might cause file name clashes.

   For example, you might use the same package name that is used for the runtime connector, such as `connector.database`.
7. In **Name**, specify a name for the node.

   The name must be one of the following values:

Connector interaction type	User-defined node name
Input	ComIbmEventInput
Output	ComIbmOutput
Request	ComIbmRequest

8. In **Display name**, enter a name for the node to include on the message flow node palette. For example, set the display name to `My Database Input`. This name is used in trace and error logs to represent the user-defined node.
9. In **Tooltip on palette**, enter some appropriate text; for example, "Input node for a database".
10. Select **Implemented in Java/C**.
11. Optional: Import an icon to represent your node.
12. Click **Finish**.

Results

A `.msgnode` file for the new user-defined node is created and is added to the project in the Application Development view. A `.properties` file of the same name is also created. The `.msgnode` file is opened in the **Message Node** editor.

What to do next

Configure the user-defined node properties for your connector by following the instructions in "Configuring the user-defined node properties and terminals for your connector" on page 2365.

Configuring the user-defined node properties and terminals for your connector

Configure the properties for your user-defined node and add input or output terminals so that you can connect it to other nodes in a message flow.

Before you begin

Create the plug-in files for your user-defined node by following the instructions in "Creating a user-defined node for a connector" on page 2364.

About this task

When you create the plug-in files for your user-defined node, a `.msgnode` file is opened in the **Message Node** editor of the Integration Development perspective. You can now add terminals and properties to the node:

- "Adding terminals to the node" on page 2365
- "Defining properties for the node" on page 2366

Adding terminals to the node

About this task

When you are creating user-defined nodes for connectors, you can add specific types of terminal according to the interaction type. User-defined nodes for connectors cannot have dynamic terminals. For

detailed information about how terminals work for different interaction types, see “User-defined node terminals for connectors” on page 2367.

Procedure

1. Open the `.msgnode` file for your user-defined node.
2. If the **Terminals** page is not already displayed, click the **Terminals** tab at the bottom of the **Message Node** editor.
3. Add an output terminal and name it "failure". The name is case-sensitive and must use lowercase characters.

   This terminal is used if the connector issues an exception at run time.
4. If you are creating an input node, delete the input terminal by selecting it, then clicking **Delete**.

   Similarly, if you are creating an output node, delete the output terminal.
5. Ensure that **Support dynamic In terminals** and **Support dynamic Out terminals** are cleared.
6. Save the file.

Defining properties for the node

About this task

A message flow developer can add a message flow node to a message flow, then customize that node by setting the available properties. You can define those properties for your user-defined node on the **Properties** tab of the Message Node editor.

For a connector, you must define a hidden property that specifies the connector provider name. You can then add any other relevant properties, such as a database name, host server name, or password. The properties that you set here must match the properties that are used by the connector at run time.

For a connector, table properties are not supported, and you cannot use a custom compiler class or a custom editor class.

Procedure

1. Open the `.msgnode` file for your user-defined node.
2. On the **Properties** tab, click the node name in the property hierarchy to view the properties that are associated with it.
3. Clear **Use integration node default values**.
4. If the node is an input node, select **Input node**.
5. By default, all properties are grouped under the Basic group. You can add new groups in which to place properties. When a flow developer selects your user-defined node in the IBM App Connect Enterprise Toolkit, each group of properties is shown on a separate tab in the **Properties** view. To create more groups of properties, click **Add Property Group**  and enter a suitable name for the group.

6. Add a hidden property to specify the provider name for the connector.
   a) Select the name of a property group, click **Add Simple Property**  , then set the name to `connectorName`.
   b) Select **Built-in** and set it to `String`.
   c) Set the default value to the name of the provider, which must match the **`connectorProvider`** name in the descriptor file of your compiled connector JAR file. This name is used for all interaction types of this particular connector (input, output, or request).
   d) Select **Hidden**.
      This option ensures that this property is associated with the connector, but the message flow developer cannot see or change the value.
   e) Because these properties are hidden, do not select **Read Only**, **Mandatory**, or **Configurable**, and do not specify field-level help.
   f) Do not set **Custom Compiler Class** or **Custom Editor Class**.
7. Add properties to be set by a message flow developer in a similar way.
   Ensure that **Hidden** is cleared for these properties so that the message flow developer can customize them.
8. Optional: Drag the properties in the properties hierarchy to change the order in which they are listed on the properties page.
9. Save and close the `.msgnode` file.
10. Optional: You can customize the text that is displayed in the node properties view for each property. When you define the property in the `.msgnode` file, the name cannot contain spaces. Therefore, you might want to edit the property name so that the name in the Properties view contains spaces.
    To set the text, open the `.properties` file for your user-defined node and edit the following line:
    ```
    Property.propertyName=Property name
    ```
11. Save and close the `.properties` file.

What to do next

Package your user-defined node by following the instructions in "Packaging the user-defined node project for a connector" on page 2368.

User-defined node terminals for connectors

The terminals that you can add to your user-defined node depend on the interaction type.

The following table describes which terminals are available for each type of connector, and how those terminals behave. You cannot use dynamic terminals on a user-defined node that represents a connector.

Interaction type	User-defined node terminals	Behavior
Input	• Output terminal • Failure terminal	The data that is passed through the output terminal includes properties and the body of the message. The properties are copied to the local environment of the message flow at this location: `LocalEnvironment > connectorName > Input > propName = propValue`

Interaction type	User-defined node terminals	Behavior

Output	• Input terminal
	• Output terminal
	• Failure terminal
		The input terminal receives the body of the message, and receives properties from the local environment at this location:
		`LocalEnvironment > Destination > connectorName > Output > propName = propValue`
		The output terminal passes the body of the message without changes. If properties are returned, they are placed in the local environment at this location:
		`LocalEnvironment > WrittenDestination > connectorName > propName = propValue`
		The failure terminal propagates the body of the message and the properties without changes.

Request and response	• Input terminal • Output terminal • Failure terminal	The input terminal receives the body of the message, and it receives properties from the local environment at this location: `LocalEnvironment > Destination > connectorName > Request > propName = propValue` The output terminal passes the body of the message that is returned by the connector. The failure terminal propagates the body of the message and the properties without changes.

Packaging the user-defined node project for a connector

Package the user-defined node project to make it available for other users.

Before you begin

Create the user-defined node for your connector in the IBM App Connect Enterprise Toolkit by following the instructions in "Developing a user-defined node to represent a connector" on page 2364.

Procedure

To package your user-defined node project, complete the following steps in the IBM App Connect Enterprise Toolkit:

1. In the Independent Resources folder of the Application Development view, right-click the user-defined node project that you want to package, then select **Package**.

   The **Package and Distribute User-Defined Nodes** wizard opens.
2. Select **Plug-in jars**, then click **Next**.
3. Select the plug-in that you want to package.

   The appropriate plug-in is selected by default.
4. Select the directory where you want to save the package, then click **Finish**.

Results

The plug-in JAR file is saved in the specified directory, within a folder called `plugins`.

What to do next

You can distribute the plug-in JAR file from your file system to message flow developers who want to use the connector in a message flow. They must install the plug-in by following the instructions in "Installing the user-defined node for a connector in the IBM App Connect Enterprise Toolkit" on page 2370.

Installing a connector in IBM App Connect Enterprise

Install the connector on the integration node on which you want to test its function.

Before you begin

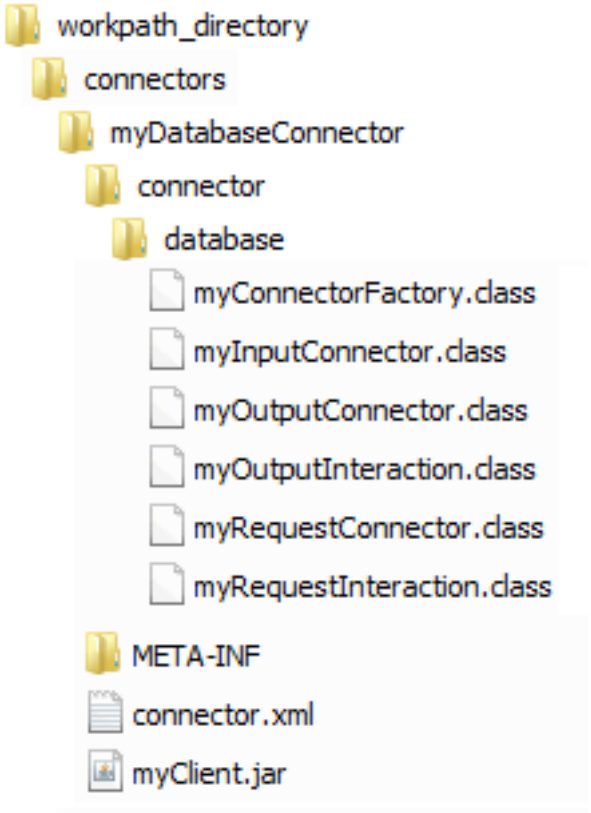
Package your connector. For more information, see "Packaging a connector for the runtime environment" on page 2362.

Procedure

To install the connector on the integration node, complete the following steps:

1. Stop the integration node on which you want to install your connector.
2. Save the JAR file that contains your connector in *workpath*`/connectors`.
   For example, the default workpath directory on Windows is `C:\ProgramData\IBM\MQSI`.

   If the connector JAR file contains other JAR files, extract the connector JAR file to a folder with the same name as the parent JAR file under *workpath*`/connectors`. For example:



3. Restart the integration node to ensure that the integration node picks up the new file.
   The connector is available to all integration servers on that integration node.

Results

The integration node loads the connector files during initialization. After the files are loaded, the integration node calls the registration functions in the connector and records what nodes or parsers the connector supports.

What to do next

Install the user-defined node for your connector by following the instructions in "Installing the user-defined node for a connector in the IBM App Connect Enterprise Toolkit" on page 2370.

Installing the user-defined node for a connector in the IBM App Connect Enterprise Toolkit

Before you can use a user-defined node in a message flow, you must install it into your IBM App Connect Enterprise Toolkit.

Before you begin

Package your user-defined node project by following the instructions in "Packaging the user-defined node project for a connector" on page 2368.

About this task

You can develop message flows by using the user-defined node in the same way as the built-in nodes. To install the user-defined node project, complete the following steps.

Procedure

1. Copy the user-defined node plug-in JAR file into the `dropins` folder in your IBM App Connect Enterprise Toolkit installation location: *`install_dir`*`/tools/dropins`.

   For example, the default installation directory on Windows is `C:\Program Files\IBM\IIB\`*`version`*`\`.
2. To ensure that the new files are available to the IBM App Connect Enterprise Toolkit, restart the Toolkit.

Results

The user-defined node for your connector is displayed in the palette in the Message Flow editor under the category that you specified for the user-defined node project.

What to do next

Test your connector in a message flow by following the instructions in "Testing a connector in IBM App Connect Enterprise" on page 2370.

Testing a connector in IBM App Connect Enterprise

To test the user-defined node that you create for a connector, add your node to a message flow. You can then use one of the debugging tools that are provided with IBM App Connect Enterprise.

Before you begin

- Complete the connector development steps for the runtime environment and IBM App Connect Enterprise Toolkit. For more information, see "Developing connectors" on page 2351.
- Install the user-defined node in the IBM App Connect Enterprise Toolkit. For more information, see "Installing the user-defined node for a connector in the IBM App Connect Enterprise Toolkit" on page 2370.

About this task

After you create a user-defined node for your connector, you can test it by adding it to a message flow in the IBM App Connect Enterprise Toolkit. You add a user-defined node for a connector in the same way that you would add any node to a message flow. You can use debugging and logging tools to ensure that the user-defined node is behaving as expected.

Procedure

To test your user-defined node in the IBM App Connect Enterprise Toolkit, complete the following steps:

1. Create a message flow by following the instructions in "Creating a message flow" on page 574.

2. Add your user-defined node to the message flow by following the instructions in "Adding a message flow node" on page 618. Add other appropriate message flow nodes to complete your message flow logic.
3. Use any of the following methods to test your message flow:

   - **Flow Exerciser**

     To check that your message flow is processing messages as expected, send messages to the flow by using the Flow Exerciser. You can then use the Flow Exerciser to show the path that each message takes, and the structure and content of the logical message tree at any point in a message flow. For more information, see "Testing your message flow by using the Flow Exerciser" on page 648.

   - **Flow debugger**

     You can set breakpoints in a message flow, then step through the flow while you examine and change message variables. To use the flow debugger, you must first deploy your message flow. For more information, see "Testing your message flow by using the flow debugger" on page 659

   - **User trace**

     To view logging output from the connector, enable user trace. Message flow nodes write messages to user trace when they are processing work. These messages show the activity in a message flow, such as which nodes are activated, what code they run, and through which terminals the messages are sent. To use user trace, you must first deploy your message flow.

     Message about your message flow are shown in the trace log file with a name in the following form: *flowName_udnDisplayName*. If the name of your user-defined node contains spaces, the spaces are replaced by underscore characters. For example, if the message flow is called `myDatabaseTestFlow.msgflow`, and the display name of the user-defined node is `My Database Input`, the entry in the trace log file is labeled `myDatabaseTestFlow_My_Database_Input`.

   - **Trace nodes**

     You can add a Trace node to a message flow to write debugging messages to a file, to user trace, or to the system log. You can then review those messages after the message flow processes some data. For more information, see "Testing your message flow by adding Trace nodes" on page 685.

Developing user-defined nodes

You can develop user-defined nodes to extend the function of IBM App Connect Enterprise.

About this task

Your can develop user-defined nodes from a subflow project, or by creating a subflow in a static library.

"Packaging a subflow as a user-defined node provides all the benefits of a subflow, including reusability and maintainability, and the following benefits:

- You can deploy a subflow without the need to compile and inline resources to produce cmf in a BAR file.
- You can use other resources, such as ESQL, maps, subflows, and message models within the function of a subflow user-defined node.
- You can use subflow user-defined nodes within subflows so that you can easily take advantage of both the subflow Use-defined node feature and features such as REST APIs and Integration Services.
- The WebUI can show the true structure of the message flow.
- The flow debugger can step into the subflows.
- The subflow is displayed in the palette in the Message Flow editor.

For more information about developing user-defined nodes, see the following topics:

- "Developing subflow user-defined nodes in a static library" on page 2372
- "Developing user-defined nodes from a subflow project" on page 2382

Developing subflow user-defined nodes in a static library

You can develop a subflow user-defined node in a static library to extend the function of IBM App Connect Enterprise.

About this task

Your installed version of IBM App Connect Enterprise must be 12.0.2.0 or later to be able to develop a subflow user-defined node in a static library.

Two roles of users exist, who can use the capability of subflow user-defined nodes that are developed in a static library: subflow user-defined node authors and subflow user-defined node consumers.

- **Subflow user-defined node authors** are developers of IBM App Connect Enterprise who want to develop, test, and package a subflow user-defined node that is used for a specific purpose. Subflow user-defined node authors do the following tasks:
  - Develop and test subflow user-defined nodes.
  - Package and distribute subflow user-defined nodes.

As a subflow user-defined node author, you develop a subflow user-defined node by first creating a subflow in a static library. Then, add nodes in the subflow so that the subflow serves a specific purpose. You then test the subflow user-defined node by adding the subflow to a message flow and testing the message flow. After you verify the behavior of the subflow user-defined node, you generate a packaged subflow user-defined library from the static library that contains the subflow user-defined node that you developed. The packaged subflow user-defined library is in the form of a `.zip` file, which you distribute to subflow user-defined node consumers who want to install your subflow user-defined node to use in message flows.

As a subflow user-defined node author, if you develop a subflow as a user-defined node within a static library, you have the following benefits:

- You can include other resources, such as ESQL, maps, subflows, and message models within the function of a subflow user-defined node.
- During development, you can use the flow debugger to step into the subflows.
- During development, you can use the IBM App Connect Enterprise web user interface to view the internal structure of the subflow.

Subflow user-defined node consumers are developers of IBM App Connect Enterprise message flows who want to use subflow user-defined nodes in their message flows.

As a subflow user-defined node consumer, if you install a subflow that is packaged as a user-defined node within a static library, you have the following benefits:

- The subflow node icons appear in the palette and you can include them in your message flows.
- The contents of the user-defined-node are hidden. The IBM App Connect Enterprise Toolkit and the IBM App Connect Enterprise web user interface display the subflow as a single node.
- You can deploy message flows that use subflow user-defined nodes as source deploy. It is not necessary to compile them inline to produce a cmf in the BAR file.
- You can use subflow user-defined nodes in subflows, which enables you to use their features, and other features such as REST APIs and integration services, in subflows.
- The contents of the user-defined subflow node are automatically packaged within the BAR file for applications that use the user-defined subflow node.

Procedure

As a subflow user-defined node author, develop, test, package, and distribute a subflow user-defined node by completing step “1” on page 2372 to step “4” on page 2373:

1. “Developing a subflow user-defined node from a subflow in a static library” on page 2373
2. “Testing a subflow user-defined node” on page 2377

3. “Packaging a subflow user-defined node by generating a packaged subflow user-defined library” on page 2379
4. “Distributing a packaged subflow user-defined library” on page 2379

As a subflow user-defined node consumer, install or uninstall a user-defined node by completing step 5 and step 6:

5. “Installing a subflow user-defined node” on page 2380
6. “Uninstalling a subflow user-defined node” on page 2381

Developing a subflow user-defined node from a subflow in a static library

Develop a subflow user-defined node by creating a subflow in a static library.

About this task

Your installed version of IBM App Connect Enterprise must be 12.0.2.0 or later to be able to create a subflow user-defined node by creating a subflow in static library.

As a user-defined subflow author, you can develop, test, package, and distribute a subflow user-defined node that was created as a subflow in static library. Using subflow user-defined nodes that are packaged in a static library provides extra flexibility for creating the subflow user-defined nodes and for how the user-defined nodes are consumed by users. As a user-defined subflow author, you must consider the following factors:

- The static library that contains a subflow user-defined node can also contain resources such as message models, schemas, ESQL, and Java classes. These resources are visible to consumers of the subflow user-defined node when they build their message flow. For example, for a subflow user-defined node that parses an input message that is based on a DFDL message model, the caller of the subflow user-defined node can validate the input message by using a "Rest Content Descriptor" node that specifies the message model from the static library before the subflow user-defined node is started.
- The static library can reference other static libraries that contain non-UDN subflows, ESQL, and schemas, but not message flows. The static library that contains a subflow user-defined node can also reference resources such as message models, non-UDN subflows, ESQL, and schemas (but not message flows) in other static libraries. When the subflow user-defined node library is packaged, the referenced libraries are automatically included in the subflow user-defined node library package.
- You cannot promote properties in subflows that are defined within the referenced library to the user-defined subflow node.
- When a subflow user-defined node uses a subflow from a referenced static library, you must avoid promoting properties from the referenced (nested) subflow to the subflow user-defined node. User-defined subflow consumers cannot set these properties.
- When you define the subflows, you must avoid defining them within the default broker schema. Use an explicit schema to avoid conflicts for consumers of the subflow user-defined nodes.
- When the application or library that uses a subflow user-defined node is packaged into a BAR file, the static library that contains the user-defined node is automatically included within the application or library within the BAR file.
- If you update the subflow user-defined node, the user-defined subflow consumer must install the new version of the user-defined node library into the toolkit and regenerate the BAR file before they can use the new version of the subflow user-defined node.
- When a subflow user-defined node is used by a message flow or subflow, a reference to the subflow user-defined node library is automatically created from the application or library that owns the message flow or subflow. If you later remove the subflow user-defined node, you must manually remove this reference if it is no longer required.

Create a subflow user-defined node by completing the following steps:

Procedure

1. Create a static library by completing the following steps:
   a) Open the **New Library** wizard by using one of the following methods in the IBM App Connect Enterprise Toolkit:
      - Click **File** > **New** > **Library**.
      - In the Application Development view, click **New**, then click **Start by creating a library**.
      - Right-click the white space in the Application Development view, then click **New** > **Library**.
   b) Enter a name for the library. For example, *DeliverFood*.
   c) Select **Static Library**.
   d) Optional: If this library is dependent upon resources in other static libraries, click **Next** and select the existing libraries that you want to reference from the list that is displayed.

      A static library can reference other static libraries only. If you add a reference to a static library that refers to other static libraries, all referenced static libraries are included.

      When you create a subflow user-defined node from a subflow in a static library, the static library must not contain message flows.
   e) Click **Finish**.

   Your new static library with the name *DeliverFood*, appears in the Application Development view.

For more information about libraries, see "Libraries" on page 1945, "Static libraries" on page 1958, and "Creating a library" on page 1964.

2. Add a subflow to the static library that you created in Step 1 by completing the following steps:
   a) Right-click your library in the Application Development view.
   b) Select **New** and then select **subflow** to open the **New Subflow** wizard.

      The name of the static library that you created in Step 1 appears in the `Library name` field.
   c) Enter a name for your subflow in the `Subflow name` field. For example, *HandleOrders*.
   d) Deselect the checkbox for **Use default broker schema**. You must not use the default broker schema when you define user-defined subflow nodes.
   e) In the **Schema** field, enter the name of the schema that you want to use. For example, *my.schema*.

      For more information about creating schemas, see "Creating a broker schema" on page 1966.
   f) Click **Finish**.

      Your new subflow appears under the static library that you created in step 1, and it has the name extension `.subflow`. In this example, the static library name is *Deliver Food*, and the subflow name is *HandleOrders.subflow*. The subflow can be deployed as source, and other subflows and libraries can be added as dependencies.

      Your new subflow also opens in the **Editor** window of the IBM App Connect Enterprise Toolkit, where you must edit it as described in step 3.
3. Add nodes in the subflow so that the subflow serves a specific purpose. From the **Palette**, add message flow nodes to the subflow, and configure and connect them as required. For more information about adding and connecting message flow nodes, see "Defining message flow content" on page 614, "Adding a message flow node" on page 618, "Configuring a message flow node" on page 624, and "Connecting message flow nodes" on page 636.
4. Define the name of the category in which the user-defined subflow nodes that are contained within the library appear on the palette, by completing the following steps:
   a) Right-click the static library to open the menu and select **Subflow user defined node** to open the submenu.
   b) Select **Set palette category** to open the **palette** wizard.
   c) By default, the name of the library is used as the name for the category. If you want to use a different name, for example *Supermarket*, edit the `Specify a category` field in the "Category

settings" section of the wizard. Alternatively, you can set the category to be one of the existing categories that are provided in the palette.

d) Set the icon that is shown in the palette for the subflow user-defined node by selecting **Import from file system** or **Import from workspace** in the "Category icons" section.

You can create icons by using a graphic design application of your own choice and storing them in a convenient location in your file system or workspace. The icon that is shown in the palette for the subflow user-defined node must be 16 x 16 pixels and be of type `.png`.

e) Click **Finish**.

5. Add the subflow to the palette category by completing the following steps:

a) In the Application Development view, right-click the subflow to open the menu and select **Subflow user defined node** to open the submenu.

b) Select **Add to palette** in the submenu to open the **Add subflow to flow editor palette** wizard.

The `Name` field in the "Category settings" section is shown as a read-only field for information only. By default, the name of the library is shown unless you changed the category name at step 4.

c) Optional: If you want the user-defined subflow node to have a different name on the palette from the name in the "Category settings" section, edit the `Name` field in the "Node settings" section.

In this example, the static library was called *Deliver Food* but the category name was changed to *Supermarket* at step 4.

d) Optional: Edit the `Tooltip` field in the "Node settings" section to set the tooltip. The tooltip is shown when you hover over the subflow user-defined node in the palette or in the flow editor.

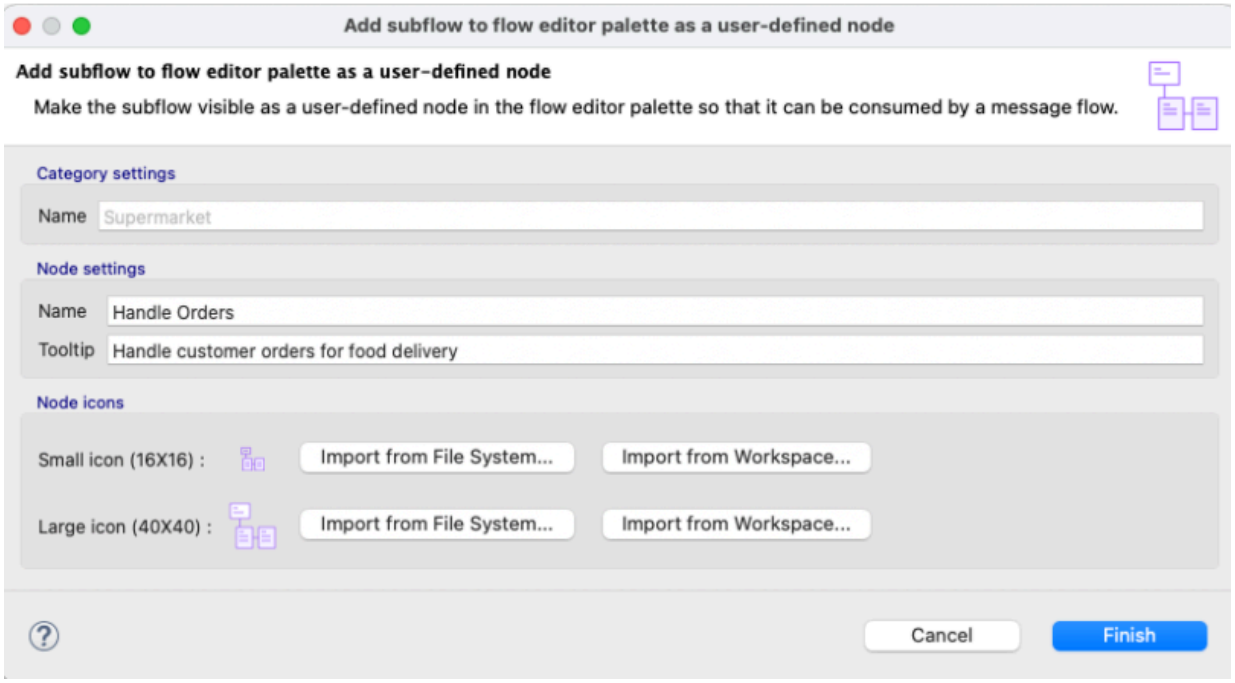
e) Set the small icon that is shown in the palette for the subflow user-defined node by selecting **Import from file system** or **Import from workspace** in the "Node icons" section.

f) Set the large icon that is shown in the flow editor for the subflow user-defined node by selecting **Import from file system** or **Import from workspace** in the "Node icons" section.

You can create icons by using a graphic design application of your own and storing them in a convenient location in your file system or workspace. The small icon that is shown choice in the palette for the subflow user-defined node must be 16 x 16 pixels and be of type `.png`. The small

large icon that is shown in the flow editor for the subflow user-defined node must be 40 x 40 pixels and be of type `.png`.

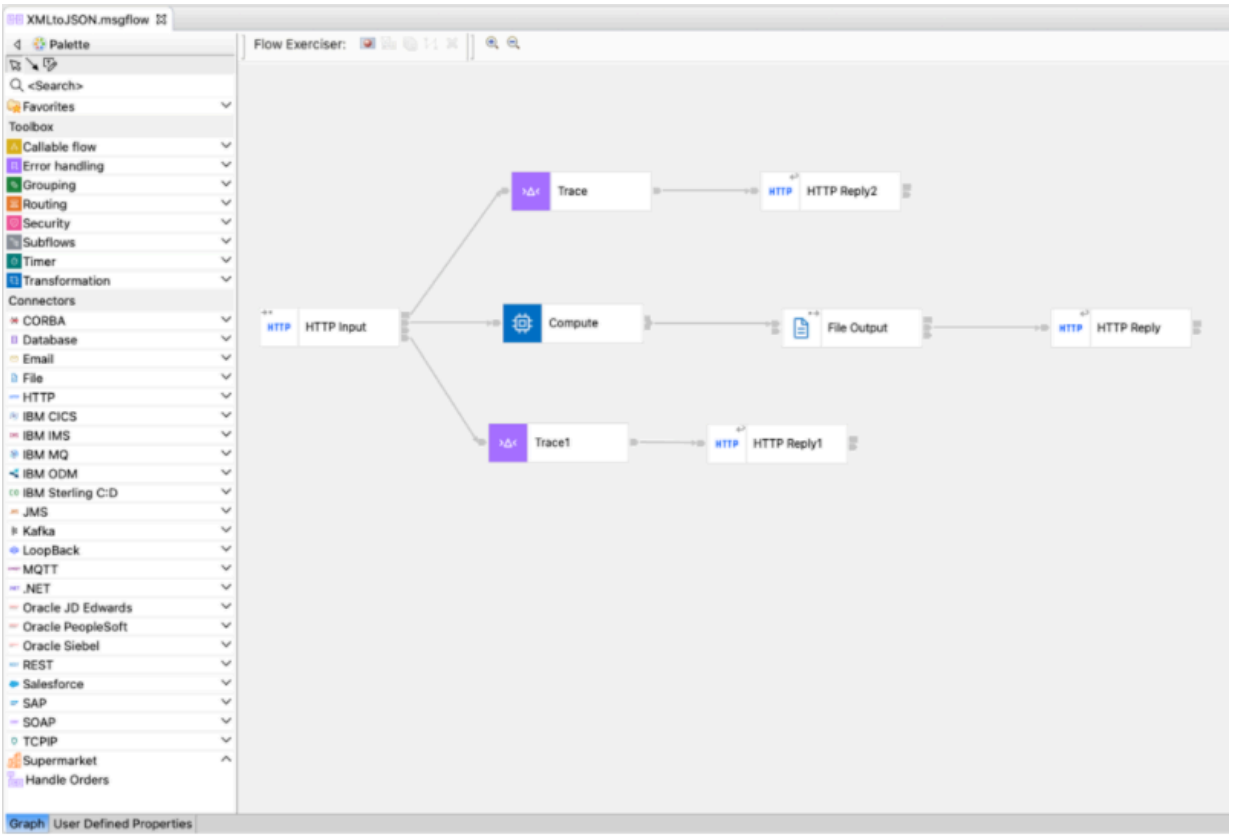
g) Click **Finish**.



Your user-defined node is added to the palette. When you next open a message flow, the new category and subflow user-defined node appear in the palette. In this example, you added a subflow that is called *HandleOrders* to the static library *DeliverFood* and used it to define a subflow user-defined node

called *HandleOrders* under the palette category *Supermarket*. The tooltip is set to *Handle Customer orders for food delivery*.

You can drag a user-defined node from the palette onto the canvas. In this environment, you can continue to update and test your user-defined subflow node in preparation for packaging the subflow user-defined node for delivery to the subflow user-defined node consumers.



What to do next

Before you package and distribute the user-defined node, you can test it as described in "Testing a subflow user-defined node" on page 2377.

Testing a subflow user-defined node

Test your subflow user-defined node before you package and distribute it.

Before you begin

Develop a subflow user-defined node from a subflow in a static library and add it to a message flow as described in "Developing a subflow user-defined node from a subflow in a static library" on page 2373.

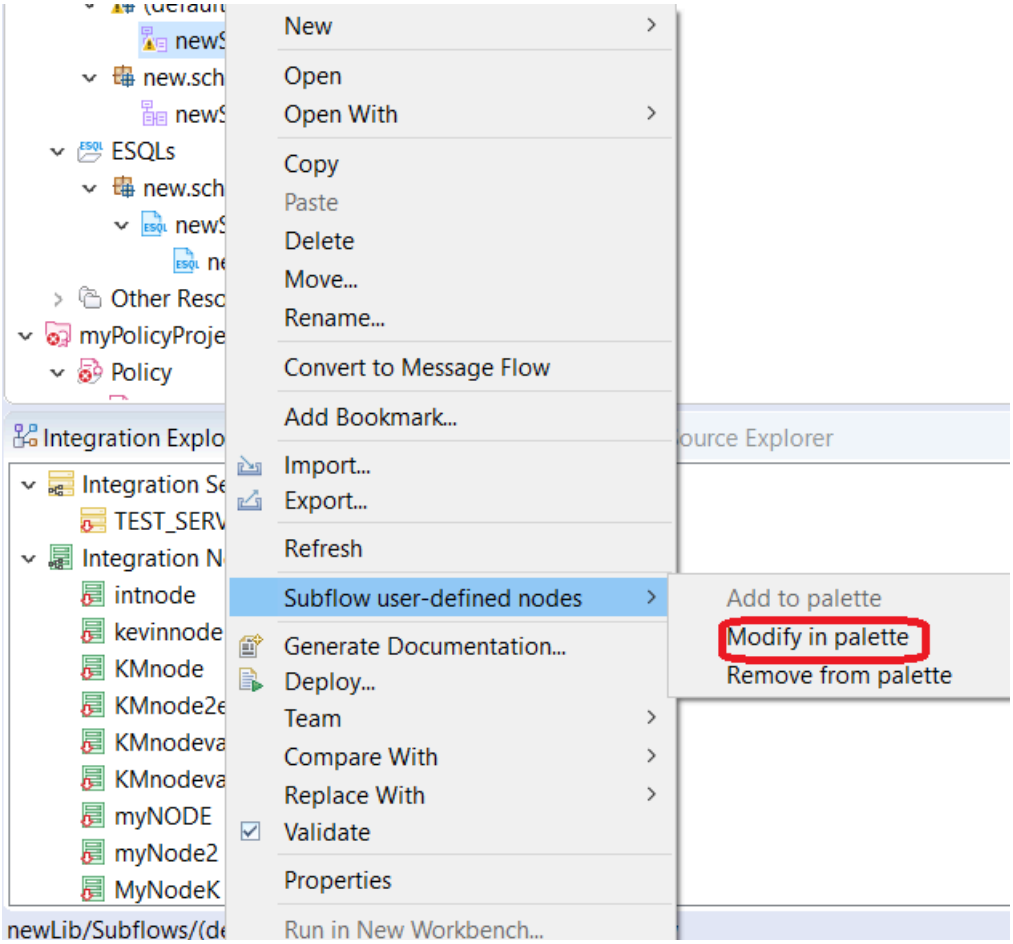
Procedure

Test your subflow user-defined node by completing the following steps:

1. Deploy, test, and if necessary, troubleshoot the message flow to which you added your subflow user-defined node. For more information about deploying and testing message flows, see How do I deploy and test ?

If you need to modify the subflow in the palette category or remove the subflow from the palette category, complete the following steps:

2. Optional: Modify the subflow in the palette category by completing the following steps:
   a) In the Application Development view, right-click the subflow to open the menu and then select **Subflow user defined node**, to open the submenu.
   b) Select **Modify in palette** in the submenu to open the **subflow** wizard.
   c) Modify the settings for the subflow user-defined node as required. If the icons are changed, they are not updated in the palette until you restart the toolkit.



3. Optional: Remove the subflow from the palette category by completing the following steps. If you attempt to delete a subflow that is being used as a subflow user-defined node, you must first use this option to remove the subflow user-defined node from the palette.
   a) In the Application Development view. Right-click the subflow to open the menu and select **Subflow user defined node** to open the submenu.
   b) Select **Remove from palette** in the submenu.

What to do next

When your node behavior is complete and correct, package and distribute the user-defined node, as described in "Packaging a subflow user-defined node by generating a packaged subflow user-defined library" on page 2379 and "Distributing a packaged subflow user-defined library" on page 2379.

Packaging a subflow user-defined node by generating a packaged subflow user-defined library

If you want to distribute your subflow user-defined node to subflow user-defined node consumers, you must package it by generating a packaged subflow user-defined library.

Before you begin

Develop and test a subflow user-defined node as described in "Developing a subflow user-defined node from a subflow in a static library" on page 2373 and "Testing a subflow user-defined node" on page 2377.

About this task

To package a subflow user-defined node by generating a subflow user-defined library, complete the following steps:

Procedure

1. In the Application Development view of the IBM App Connect Enterprise Toolkit, right-click the static library that contains the subflow user-defined node that you want to package. When the menu opens, select **Subflow user defined nodes** to open the submenu.
2. Click **Package subflow user defined nodes** to open the packaging wizard.
3. Enter the name of the .zip file that will contain the packaged subflow user-defined nodes. By default, the library name is used with the extension `.zip`. For example, for the library name *DeliverFood*, the name *DeliverFood.zip* is used as the name of the .zip file.
4. Enter the location of the packaged subflow user-defined library. For example, *C:\myzips*.
5. Click **Finish** to create the .zip file.

   The .zip file is saved to the location that you specified.

What to do next

You can now install the subflow user-defined node, or distribute the packaged subflow user-defined library to enable subflow user-defined node consumers to install the subflow user-defined node. For more information about installing and distributing subflow user-defined nodes, see "Installing a subflow user-defined node" on page 2380 and "Distributing a packaged subflow user-defined library" on page 2379.

Distributing a packaged subflow user-defined library

Distribute the packaged subflow user-defined library to make the subflow user-defined node available for use in a message flow.

About this task

You can distribute the packaged subflow user-defined library by copying the `.zip` file from your file system into a common drive where it can be accessed by subflow user-defined node consumers. Alternatively, you can send it as an email attachment to subflow user-defined node consumers who can download it into their file system.

What to do next

Subflow user-defined node consumers can now install the subflow user-defined node as described in "Installing a subflow user-defined node" on page 2380.

Installing a subflow user-defined node

Install a subflow user-defined node for use to develop message flows.

Before you begin

You can develop message flows by using subflow user-defined nodes in the same way as the built-in nodes. Before you begin, you must develop, test, and package a subflow user-defined node as described in Developing a subflow user-defined node from a subflow in a static library, "Testing a subflow user-defined node" on page 2377, and Packaging a subflow user-defined node by generating a packaged subflow user-defined library. You must then install the static library that contains the subflow user-defined node into your IBM App Connect Enterprise Toolkit before you can use the subflow user-defined node in a message flow.

If you install a subflow user-defined library, you cannot create a project in the workspace that has the same name as the installed library. An error is shown if you attempt to create a project that has the same name as the installed project.

Procedure

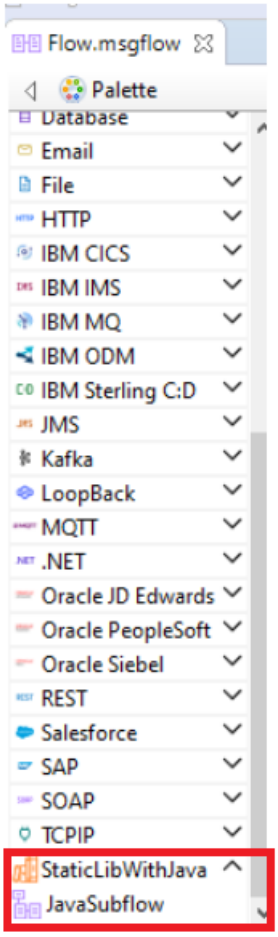
1. In the IBM App Connect Enterprise Toolkit, click **Project**, and then select **Manage subflow user-defined libraries** from the menu.

   The Manage subflow user-defined Libraries dialog opens. From this dialog, you can see a list of the installed libraries and install or uninstall a library.
2. Click **Install** to open a dialog where you can use **Windows Explorer** to find the package `.zip` file that you want to install.
3. Select the package `.zip` file that you want to install.
4. Click **Open** to open a dialog that displays details of the library contents.

5. Click **Yes** to confirm that you want to install the library.

   The library is installed. The currently installed library list is updated with the new library.

   The subflow now appears in the palette when you open a message flow. In this example, the subflow is *JavaSubflow* under the library *StaticLibWithJava*.



   You can drag the node onto the canvas. When you package the application or library that uses a subflow user-defined node into a BAR file, the static library that contains the user-defined node is automatically included within the application or library within the BAR file.

6. If a subflow user-defined node is updated, you must install the new version of the user-defined node library into the toolkit. You must regenerate the BAR file before you can use the updated version.

What to do next

When you no longer require the subflow user-defined node, you can uninstall it. For more information, see "Uninstalling a subflow user-defined node" on page 2381.

Uninstalling a subflow user-defined node

Uninstall a packaged subflow user-defined library that was used to develop message flows.

About this task

If you installed a packaged subflow user-defined library that you no longer require, you can uninstall it by completing the following steps:

Procedure

1. In the IBM App Connect Enterprise Toolkit, click **Project** and then select **Manage subflow user-defined libraries** from the menu.

   The Manage subflow user-defined libraries dialog is displayed, containing a list of all subflow user-defined libraries that have been installed.
2. Select the library that you want to uninstall.
3. Click **Uninstall**.

   A dialog opens where you are asked to confirm that you want to uninstall the subflow user-defined library.
4. Click **Yes**.

   The library is removed, and the list of currently installed subflow user-defined libraries is updated.

   The library is no longer available in the workspace, and any flows that use the library now display errors.
5. When a subflow user-defined node is used by a message flow or subflow, a reference to the subflow user-defined node library is automatically created from the application or library that owns the message flow or subflow. If you later remove the subflow user-defined node, you must manually remove this reference.

Developing user-defined nodes from a subflow project

You can develop a user-defined node to extend the function of IBM App Connect Enterprise.

Before you begin

Read the following topics:

- "User-defined extensions overview" on page 2321
- "Why use a user-defined extension?" on page 2322
- "User-defined nodes" on page 2331
- "Which type of user-defined extension to use" on page 2331

About this task

Consider the following restrictions and factors when you develop user-defined nodes:

- Interfacing a C user-defined node to Java and providing a JNI wrapper is not supported. This restriction exists because the integration node internally initializes a JVM, which is not available through the user-defined extension interface. The JVM initializes with various parameters that are specific to the requirements of the integration node. Because only one JVM exists in a process, whoever initializes it first specifies these parameters. If a user-defined node uses Java, and the integration node is initialized first, these parameters might not be suitable for the user-defined node. If the user-defined node creates the JVM before the integration node starts, the integration node might not function correctly.
- User-defined input nodes can support only XML, BLOB, and the WebSphere MQ parsers.
- Avoid functions that are specific to an operating system. If you code in this way, your user-defined extensions can work on various operating systems without requiring changes to the source code.
- Always put a user-defined node into a non-default schema because a user-defined node in a broker schema is known to other message flows by its schema qualified name. For example, if a user-defined node is named `ErrorHandler` and it is in broker schema `com.ibm.mb.toolkit`, it is referenced as `com.ibm.mb.toolkit.ErrorHandler`. If a second provider also has an error handler that is named `ErrorHandler` and it is in broker schema `com.`*`xxx`*`.product`, it is referenced as `com.`*`xxx`*`.product.ErrorHandler`. A user-defined node in a default schema is addressed by its name only. Therefore, if two different providers develop two unrelated error handlers and both are named `ErrorHandler` and both are placed in a default schema, when both user-defined nodes are in the plug-in space for a third user, the reference to `ErrorHandler` is ambiguous.

- If you want to use a subflow to create a user-defined node, read the limitations section in the following topic: "Using a subflow as a user-defined node" on page 2344.

To implement a user-defined node, complete the following tasks in the specified order:

Procedure

1. "Designing a user-defined node" on page 2383
2. "Creating a user-defined node" on page 2383
3. "Packaging and distributing user-defined extensions" on page 2449
4. "Testing a user-defined node" on page 2430
5. "Packaging and distributing a user-defined node project" on page 2452

Designing a user-defined node

Decide what type of node you need to implement the functions that are required by your application.

Before you begin

Read "Deciding which nodes to use" on page 585 to understand the different types of node. You might need more than one node to implement all the functions that you require.

About this task

The functions that you require might not be satisfied by a template that already exists for several reasons:

- The functions that you require do not relate to interacting with external systems. Most of the node design pattern concentrates on communication with external systems, which is the most likely requirement for a user-defined node.
- The functions that are required are not well suited to the IBM App Connect Enterprise architecture, so you should implement them in an end application, or an application server.
- The functions require complex control and state information, which you should not implement as a plug-in.

Creating a user-defined node

You can write user-defined nodes in C, Java, or from a subflow.

Before you begin

Read "Designing a user-defined node" on page 2383.

About this task

When you have created a user-defined node, you can test it, as described in "Testing a user-defined node" on page 2430. If you want to test or use user-defined nodes or parsers on multiple computers, follow the instructions given in "Packaging and distributing user-defined extensions" on page 2449.

Decide whether you want to create a user-defined node in C, Java, or from a subflow, then follow the instructions in the appropriate topic.

- "Creating a user-defined extension in C" on page 2384
- "Creating a user-defined extension in Java" on page 2403
- "Creating a user-defined node from a subflow" on page 2419
- "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421

The following table shows the tasks that are involved in creating the different types of user-defined node.

Objective	Tasks to complete
To create your own Java node by using the IBM App Connect Enterprise Toolkit	1. "Creating an input node in Java" on page 2403 or "Creating a message processing or output node in Java" on page 2409 2. "Using error logging from a user-defined extension" on page 2458 3. "Compiling a Java user-defined node" on page 2418 4. "Testing a user-defined node" on page 2430 5. "Packaging and distributing a user-defined node project" on page 2452 6. Installing a user-defined extension to current and past versions of
To create your own C node:	1. "Creating an input node in C" on page 2385 or "Creating a message processing or output node in C" on page 2392 2. "Using error logging from a user-defined extension" on page 2458 3. "Compiling a C user-defined extension" on page 2400 4. "Installing user-defined extension runtime files on an integration node" on page 2455 5. "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421 6. "Testing a user-defined node" on page 2430 7. "Packaging and distributing a user-defined node project" on page 2452 8. Installing a user-defined extension to current and past versions of
To create your own node from a subflow	1. "Creating a user-defined node from a subflow" on page 2419 2. "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421 3. "Testing a subflow user-defined node project" on page 2433 4. "Packaging and distributing a user-defined node project" on page 2452 5. "Installing a user-defined node" on page 620

Creating a user-defined extension in C
You must complete a series of tasks to create user-defined extensions that use the C language.

About this task

You can write user-defined nodes and user-defined parsers in C.

Complete the appropriate tasks from the following list:

- "Creating an input node in C" on page 2385
- "Creating a message processing or output node in C" on page 2392
- "Developing user-defined parsers" on page 2434
- "Compiling a C user-defined extension" on page 2400

What to do next

When you have completed this set of tasks, continue with the following tasks:

- If you have compiled a user-defined node, "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421

- “Testing a user-defined node” on page 2430
- “Packaging and distributing user-defined extensions” on page 2449

Creating an input node in C
Create a user-defined input node in C to receive messages into a message flow.

Before you begin

Read the following topics:

- “Why use a user-defined extension?” on page 2322
- “User-defined input nodes” on page 2332

About this task

A loadable implementation library, or *LIL*, is the implementation module for a C node. A LIL is implemented as a shared or dynamic link library (DLL), but has the file extension `.lil` not `.dll`.

The implementation functions that you write for the node are listed in C node implementation functions. You can call utility functions, implemented in the runtime integration node, to help with the node operation; these functions are listed in C node utility functions.

IBM App Connect Enterprise provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them.

To create an input node in C:

1. “Declaring and defining the node” on page 2385
2. “Creating an instance of the node” on page 2386
3. “Setting attributes” on page 2387
4. “Implementing the node functionality” on page 2388
5. “Overriding the default message parser attributes (optional)” on page 2388
6. “Deleting an instance of the node” on page 2390

Declaring and defining the node

About this task

To declare and define a user-defined node to the integration node, include an initialization function, `bipGetMessageflowNodeFactory`, in your LIL. The following steps outline how the integration node calls your initialization function, and how your initialization function declares and defines the user-defined node:

Procedure

1. The initialization function, `bipGetMessageflowNodeFactory`, is called by the integration node after the operating system has loaded and initialized the LIL. The integration node calls this function to understand what your LIL can do and how the integration node can call the LIL. For example:

   ```
   CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
   bipGetMessageflowNodeFactory()
   ```

2. The `bipGetMessageflowNodeFactory` function must call the utility function `cniCreateNodeFactory`. This function passes back a unique factory name (or group name) for all the nodes that your LIL supports. Every factory name (or group name) that is passed back must be unique throughout all the LILs in a single runtime integration node.

3. The LIL must call the utility function `cniDefineNodeClass` to pass the unique name of each node, and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single node called InputxNode:

```
{
 CciFactory* factoryObject;
 int rc = 0;
 CciChar factoryName[] = L"MyNodeFactory";
 CCI_EXCEPTION_ST exception_st;

 /* Create the Node Factory for this node */
 factoryObject = cniCreateNodeFactory(0, factoryName);
 if (factoryObject == CCI_NULL_ADDR) {

  /* Any local error handling can go here */
 }
 else {
  /* Define the nodes supported by this factory */
  static CNI_VFT vftable = {CNI_VFT_DEFAULT};

 /* Setup function table with pointers to node implementation functions */
 vftable.iFpCreateNodeContext = _createNodeContext;
 vftable.iFpDeleteNodeContext = _deleteNodeContext;
 vftable.iFpGetAttributeName2 = _getAttributeName2;
 vftable.iFpSetAttribute      = _setAttribute;
 vftable.iFpGetAttribute2     = _getAttribute2;
 vftable.iFpRun               = _run;

 cniDefineNodeClass(0, factoryObject, L"InputxNode", &vftable);
 }

 /* Return address of this factory object to the integration node */
 return(factoryObject);
}
```

   A user-defined node identifies itself as providing the features of an input node by implementing the `cniRun` implementation function.

   User-defined nodes have to implement either a `cniRun` or a `cniEvaluate` implementation function. If they do not, the integration node does not load the user-defined node, and the `cniDefineNodeClass` utility function fails, returning CCI_MISSING_IMPL_FUNCTION.

   For example:

```
int cniRun(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
){
  ...
  /* Get data from external source */
  return CCI_SUCCESS_CONTINUE;
}
```

   Use the return value periodically to return control to the integration node.

   When a message flow that contains a user-defined input node is deployed successfully, the `cniRun` function of the node is called repeatedly to enable the node to retrieve messages and propagate them to the rest of the message flow.

   For the minimum code required to compile a C user-defined node, see the C skeleton code.

Creating an instance of the node

About this task

To instantiate your node:

Procedure

1. When the integration node has received the table of function pointers, it calls the function `cniCreateNodeContext` for each instantiation of the user-defined node. For example, if three message flows are using your user-defined node, your `cniCreateNodeContext` function is called for each of them. This function must allocate memory for that instantiation of the user-defined node to hold the values for the configured attributes. For example:

   a) Call the `cniCreateNodeContext` function:

   ```
   CciContext* _createNodeContext(
     CciFactory* factoryObject,
     CciChar*    nodeName,
     CciNode*    nodeObject
   ){
     static char* functionName = (char *)"_createNodeContext()";
     NODE_CONTEXT_ST* p;
     CciChar          buffer[256];
   ```

   b) Allocate a pointer to the local context and clear the context area:

   ```
     p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));

     if (p) {
        memset(p, 0, sizeof(NODE_CONTEXT_ST));
   ```

   c) Save the node object pointer in the context:

   ```
      p->nodeObject = nodeObject;
   ```

   d) Save the node name:

   ```
    CciCharNCpy((CciChar*)&p->nodeName, nodeName, MAX_NODE_NAME_LEN);
   ```

   e) Return the node context:

   ```
   return (CciContext*) p;
   ```

2. An input node has a number of output terminals associated with it, but typically does not have any input terminals. Use the utility function `cniCreateOutputTerminal` to add output terminals to an input node when the node is instantiated. These functions must be invoked within the `cniCreateNodeContext` implementation function. For example, to define an input node with three output terminals:

   ```
   {
      const CciChar* ucsOut = CciString("out", BIP_DEF_COMP_CCSID) ;
      insOutputTerminalListEntry(p, (CciChar*)ucsOut);
      free((void *)ucsOut) ;
    }
    {
      const CciChar* ucsFailure = CciString("failure", BIP_DEF_COMP_CCSID) ;
      insOutputTerminalListEntry(p, (CciChar*)ucsFailure);
      free((void *)ucsFailure) ;
    }
    {
      const CciChar* ucsCatch = CciString("catch", BIP_DEF_COMP_CCSID) ;
      insOutputTerminalListEntry(p, (CciChar*)ucsCatch);
      free((void *)ucsCatch) ;    }
   ```

   For the minimum code required to compile a C user-defined node, see C skeleton code.

Setting attributes

About this task

Attributes are set whenever you start the integration node, or when you redeploy the message flow with new values.

Following the creation of output terminals, the integration node calls the `cniSetAttribute` function to pass the values for the configured attributes of the user-defined node. For example:

```
    {
      const CciChar* ucsAttr = CciString("nodeTraceSetting", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_INTEGER);
      _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constZero);
      free((void *)ucsAttr) ;
    }
    {
      const CciChar* ucsAttr = CciString("nodeTraceOutfile", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_STRING);
      _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constSwitchTraceLocation);
      free((void *)ucsAttr) ;
    }
```

The number of configuration attributes that a node can have is unlimited. However, a user-defined node must not implement an attribute that is already implemented as a base configuration attribute. The base attributes are:

- label
- userTraceLevel
- traceLevel
- userTraceFilter
- traceFilter

Implementing the node functionality

About this task

When the integration node knows that it has an input node, it calls the `cniRun` function of this node at regular intervals. The `cniRun` function must then decide what course of action it must take. If data is available for processing, the `cniRun` function can propagate the message. If no data is available for processing, the `cniRun` function must return with CCI_TIMEOUT so that the integration node can continue configuration changes.

For example, to configure the node to call `cniDispatchThread` and process the message, or return with CCI_TIMEOUT:

```
If ( anything to do )
 CniDispatchThread;

   /* do the work */

 If ( work done O.K.)
  Return CCI_SUCCESS_CONTINUE;
 Else
  Return CCI_FAILURE_CONTINUE;
Else
  Return CCI_TIMEOUT;
```

Overriding the default message parser attributes (optional)

About this task

An input node implementation typically determines what message parser initially parses an input message. For example, the MQInput node dictates that an MQMD parser is required to parse the MQMD header. A user-defined input node can select an appropriate header or message parser, and the mode in which the parsing is controlled, by using or overriding the following attributes that are included as default:

rootParserClassName
: Defines the name of the root parser that parses message formats that are supported by the user-defined input node. It defaults to `GenericRoot`, a supplied root parser that causes the integration node to allocate and chain parsers together. It is unlikely that a node would need to modify this attribute value.

firstParserClassName
Defines the name of the first parser, in what might be a chain of parsers that are responsible for parsing the bit stream. It defaults to XML.

messageDomainProperty
An optional attribute that defines the name of the message parser that is required to parse the input message. The supported values are the same as the values that are supported by the MQInput node. (See node for more information.)

messageSetProperty
An optional attribute that defines the message set identifier, or the message set name, in the `Message Set` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageTypeProperty
An optional attribute that defines the identifier of the message in the `MessageType` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageFormatProperty
An optional attribute that defines the format of the message in the `Message Format` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

If you have written a user-defined input node that always begins with data of a known structure, you can ensure that a specific parser handles the start of the data. For example, the MQInput node reads data only from IBM MQ queues, therefore this data always has an MQMD at the beginning, and the MQInput node sets firstParserClassName to MQHMD. If your user-defined node always handles data that begins with a structure that can be parsed by a specific parser, for example "MYPARSER", set firstParserClassName to MYPARSER in the following way::

1. Declare the implementation functions:

```
CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix bipGetMessageflowNodeFactory()
{
  ....
  CciFactory*      factoryObject;
  ....
  factoryObject = cniCreateNodeFactory(0, (unsigned short *)constPluginNodeFactory);
  ...
  vftable.iFpCreateNodeContext = _createNodeContext;
  vftable.iFpSetAttribute      = _setAttribute;
  vftable.iFpGetAttribute      = _getAttribute;
  ...
  cniDefineNodeClass(&rc, factoryObject, (CciChar*)constSwitchNode, &vftable);
  ...
  return(factoryObject);
}
```

2. Set the attribute in the `cniCreateNodeContext` implementation function:

```
CciContext* _createNodeContext(
  CciFactory* factoryObject,
  CciChar*    nodeName,
  CciNode*    nodeObject
){
  NODE_CONTEXT_ST* p;
  ...

    /* Allocate a pointer to the local context */
    p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));
    /* Create attributes and set default values */
    {
      const CciChar* ucsAttrName  = CciString("firstParserClassName", BIP_DEF_COMP_CCSID);
      const CciChar* ucsAttrValue = CciString("MYPARSER", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttrName, CNI_TYPE_INTEGER);
      /*see sample BipSampPluginNode.c for implementation of insAttrTblEntry*/

      _setAttribute(p, (CciChar*)ucsAttrName, (CciChar*)ucsAttrValue);
      free((void *)ucsAttrName) ;
      free((void *)ucsAttrValue) ;
```

```
    }
```

In the code example above, the `insAttrTblEntry` method is called. This method is declared in the SwitchNode and TransformNode sample user-defined nodes.

Deleting an instance of the node

About this task

Nodes are destroyed when a message flow is redeployed, or when the integration server process is stopped (using the **mqsistop** command). When a node is destroyed, you must call the `cniDeleteNodeContext` function to free all used memory and release all held resources. For example:

```
void _deleteNodeContext(
  CciContext* context
){
  static char* functionName = (char *)"_deleteNodeContext()";

  return;
}
```

Extending the capability of a C input node
When you have created a user-defined node, you can extend its capability.

Before you begin

Read "Creating an input node in C" on page 2385.

About this task

After you have created a user-defined node, the following options are available:

1. "Receiving external data into a buffer" on page 2390
2. "Controlling threading and transactions" on page 2391
3. "Propagating the message" on page 2392

Receiving external data into a buffer

About this task

An input node can receive data from any type of external source, such as a file system or FTP connection, provided that the output from the node is in the correct format. For connections to queues or databases, use the built-in nodes and the API calls supplied, principally because the built-in nodes are already set up for error handling. Do not use the MQGET or MQPUT calls for direct access to IBM MQ queues.

You must provide an input buffer (or bit stream) to contain input data, and associate it with a message object. In the C API, the buffer is attached to the CciMessage object that represents the input message by using the `cniSetInputBuffer` utility function. For example:

```
{
  static char* functionName = (char *)"_Input_run()";
  void*        buffer;
  CciTerminal* terminalObject;
  int          buflen = 4096;
  int          rc = CCI_SUCCESS;
  int          rcDispatch = CCI_SUCCESS;

  buffer = readFromDevice(&buflen);
  cniSetInputBuffer(&rc, message, buffer, buflen);
}
/*propagate etc*/
```

About this task

An input node must perform appropriate end-of-message processing when a message has been propagated through a message flow. Specifically, the input node needs to cause any transactions to be committed or rolled back, and return threads to the thread pool.

Each message flow thread is allocated from a pool of threads that is maintained for each message flow, and starts execution in the `cniRun` function. You determine the behavior of a thread using the `cniDispatchThread` utility function, together with the appropriate return value.

From the `cniRun` function, you can call the `cniDispatchThread` utility function to cause another thread to start executing the `cniRun` function. The most appropriate time to execute another thread is directly after you have established that data could be available for the function to process on the new thread.

The term *transaction* is used generically to describe either a globally coordinated transaction, or a transaction controlled by an integration node. Globally coordinated transactions are coordinated by either IBM MQ as an XA-compliant Transaction Manager, or Resource Recovery Service (RRS) on z/OS. IBM App Connect Enterprise controls transactions by committing (or rolling back) any database resources, and then committing any IBM MQ units of work. However, if a user-defined node is used, the integration node cannot automatically commit any resource updates. The user-defined node uses return values to indicate whether a transaction has been successful, and to control whether transactions are committed or rolled-back. The integration node infrastructure catches any unhandled exceptions, and rolls back the transaction.

The following table describes each of the supported return values, the effect that each one has on any transactions, and what the integration node does with the current thread.

Return value	**Affect on transaction**	**Integration node action on the thread**
CCI_SUCCESS_CONTINUE	Committed	Calls the same thread again in the `cniRun` function.
CCI_SUCCESS_RETURN	Committed	Returns the thread to the thread pool.
CCI_FAILURE_CONTINUE	Rolled back	Calls the same thread again in the `cniRun` function.
CCI_FAILURE_RETURN	Rolled back	Returns the thread to the thread pool.
CCI_TIMEOUT	Not applicable. The function periodically times out while it is waiting for an input message.	Calls the same thread again in the `cniRun` function.

The following code is an example of using the SUCCESS_RETURN return code with the `cniDispatchThread` function:

```
{
  ...
  cniDispatchThread(&rcDispatch, ((NODE_CONTEXT_ST *)context)->nodeObject);
  ...
  if (rcDispatch == CCI_NO_THREADS_AVAILABLE) return CCI_SUCCESS_CONTINUE;
  else return CCI_SUCCESS_RETURN;
}
```

Propagating the message

About this task

Before you propagate a message, decide what message flow data you want to propagate, and which terminal is to receive the data.

The terminalObject is derived from a list that the user-defined node maintains.

For example, to propagate the message to the output terminal, use the `cniPropagate` function:

```
if (terminalObject) {
  if (cniIsTerminalAttached(&rc, terminalObject)) {
    if (rc == CCI_SUCCESS) {
      cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, message);
    }
  }
```

In the above example, the `cniIsTerminalAttached` function is used to test whether the message can be propagated to the specified terminal. If you do not use the `cniIsTerminalAttached` function, and the terminal is not attached to another node by a connector, the message is not propagated and no warning message is returned. Use the `cniIsTerminalAttached` function to prevent this error occurring.

Creating a message processing or output node in C
A message processing node is used to process a message in some way, and an output node is used to produce a message as a bit stream.

Before you begin

Read the following topics:

- "Why use a user-defined extension?" on page 2322
- "User-defined message processing nodes" on page 2336
- "User-defined output nodes" on page 2343

About this task

When you code a message processing node or an output node, the nodes provide essentially the same services. You can perform message processing in an output node, and you can send a message to a bit stream by using a message processing node. For simplicity, this topic refers mainly to the node as a message processing node but it does also contain information about the functions of both types of node.

A loadable implementation library (LIL), is the implementation module for a C node. A LIL is implemented as a shared or dynamic link library (DLL), but has the file extension `.lil` not `.dll`.

For more information about the C node implementation functions that you write for the node, see C node implementation functions. You can call C node utility functions, implemented in the runtime integration node, to help with the node operation; see C node utility functions.

IBM App Connect Enterprise provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them.

To create either type of node complete the following tasks:

1. "Declaring and defining your node" on page 2393
2. "Creating an instance of the node" on page 2394
3. "Setting attributes" on page 2396
4. "Implementing the node functionality" on page 2397
5. "Deleting an instance of the node" on page 2397

About this task

To declare and define a user-defined node to the integration node, include an initialization function, `bipGetMessageflowNodeFactory`, in your LIL. The following steps take place on the configuration thread and outline how the integration node calls your initialization function and how your initialization function declares and defines the user-defined node:

Procedure

1. The integration node calls the initialization function `bipGetMessageflowNodeFactory` after the operating system has loaded and initialized the LIL. The integration node calls this function to understand what your LIL can do and how the integration node can call the LIL. For example:

```
CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
bipGetMessageflowNodeFactory()
```

2. The `bipGetMessageflowNodeFactory` function must call the utility function `cniCreateNodeFactory`. This function passes back a factory name (or group name) for all the nodes that your LIL supports. The factory name (or group name) must be unique throughout all the LILs in a single runtime integration node.
3. The LIL must call the utility function `cniDefineNodeClass` to pass the unique name of each node and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single node called MessageProcessingxNode:

```
{
 CciFactory* factoryObject;
 int rc = 0;
 CciChar factoryName[] = L"MyNodeFactory";
 CCI_EXCEPTION_ST exception_st;

 /* Create the Node Factory for this node */
 factoryObject = cniCreateNodeFactory(0, factoryName);
 if (factoryObject == CCI_NULL_ADDR) {
  /* Any local error handling can go here */
 }
 else {
  /* Define the nodes supported by this factory */
 static CNI_VFT vftable = {CNI_VFT_DEFAULT};

 /* Setup function table with pointers to node implementation functions */
 vftable.iFpCreateNodeContext = _createNodeContext;
 vftable.iFpDeleteNodeContext = _deleteNodeContext;
 vftable.iFpGetAttributeName2 = _getAttributeName2;
 vftable.iFpSetAttribute      = _setAttribute;
 vftable.iFpGetAttribute2     = _getAttribute2;
 vftable.iFpEvaluate          = _evaluate;

 cniDefineNodeClass(0, factoryObject, L"MessageProcessingxNode", &vftable);

 }

 /* Return address of this factory object to the integration node */
 return(factoryObject);
}
```

   A user-defined node identifies itself as a message processing or output node by implementing the `cniEvaluate` function. User-defined nodes must implement either a `cniEvaluate` or a `cniRun`

implementation function, otherwise the integration node does not load the user-defined node, and the `cniDefineNodeClass` utility function fails, returning CCI_MISSING_IMPL_FUNCTION.

When a message flow containing a user-defined message processing node is deployed successfully, the node's `cniEvaluate` function is called for each message propagated to the node.

Message flow data is received at the input terminal of the node, that is, the message, Environment, LocalEnvironment, and ExceptionList.

For example:

```
void cniEvaluate(
  CciContext* context,
  CciMessage* localEnvironment,
  CciMessage* exceptionList,
  CciMessage* message
){
  ...
}
```

For the minimum code required to compile a C user-defined node, see C skeleton code.

Creating an instance of the node

About this task

To instantiate your node:

Procedure

1. When the integration node has received the table of function pointers, it calls the function `cniCreateNodeContext` for each instantiation of the user-defined node. For example, if three message flows are using your user-defined node, your `cniCreateNodeContext` function is called for each of them. This function allocates memory for that instantiation of the user-defined node to hold the values for the configured attributes. For example:
   a) The user function `cniCreateNodeContext` is called:

   ```
   CciContext* _Switch_createNodeContext(
     CciFactory* factoryObject,
     CciChar*    nodeName,
     CciNode*    nodeObject
   ){
     static char* functionName = (char *)"_Switch_createNodeContext()";
     NODE_CONTEXT_ST* p;
     CciChar          buffer[256];
   ```

   b) Allocate a pointer to the local context and clear the context area:

   ```
     p = (NODE_CONTEXT_ST *)malloc(sizeof(NODE_CONTEXT_ST));

     if (p) {
        memset(p, 0, sizeof(NODE_CONTEXT_ST));
   ```

   c) Save the node object pointer in the context:

   ```
      p->nodeObject = nodeObject;
   ```

   d) Save the node name:

   ```
    CciCharNCpy((CciChar*)&p->nodeName, nodeName, MAX_NODE_NAME_LEN);
   ```

   e) Return the node context:

   ```
   return (CciContext*) p;
   ```

2. The integration node calls the appropriate utility functions to find out about the node's input terminals and output terminals. A node has a number of input terminals and output terminals associated with it. Within the user function `cniCreateNodeContext`, calls must be made to `cniCreateInputTerminal` and `cniCreateOutputTerminal` to define the user node's terminals. These functions must be started within the `cniCreateNodeContext` implementation function.

   For example, to define a node with one input terminal and two output terminals:

```
{
  const CciChar* ucsIn = CciString("in", BIP_DEF_COMP_CCSID) ;
  insInputTerminalListEntry(p, (CciChar*)ucsIn);
  free((void *)ucsIn) ;
}
{
  const CciChar* ucsOut = CciString("out", BIP_DEF_COMP_CCSID) ;
  insOutputTerminalListEntry(p, (CciChar*)ucsOut);
  free((void *)ucsOut) ;
}
{
  const CciChar* ucsFailure = CciString("failure", BIP_DEF_COMP_CCSID) ;
  insOutputTerminalListEntry(p, (CciChar*)ucsFailure);
  free((void *)ucsFailure) ;
}
```

   The previous code starts the `insInputTerminalListEntry` and `insOutputTerminalListEntry` functions. You can find these functions in the sample code `Common.c`; see Sample node files. These functions define the terminals to the integration node and store handles to the terminals. Handles are stored in the structure referenced by the value returned in `CciContext*`. The node can then access the terminal handles from within the other implementation functions (for example `CciEvaluate`) because `CciContext` is passed to those implementation functions.

   The following code shows the definition of `insInputTerminalListEntry`:

```
TERMINAL_LIST_ENTRY *insInputTerminalListEntry(
  NODE_CONTEXT_ST* context,
  CciChar*         terminalName
){
  static char* functionName = (char *)"insInputTerminalListEntry()";
  TERMINAL_LIST_ENTRY* entry;
  int                  rc;

  entry = (TERMINAL_LIST_ENTRY *)malloc(sizeof(TERMINAL_LIST_ENTRY));
  if (entry) {

    /* This entry is the current end of the list */
    entry->next = 0;

    /* Store the terminal name */
    CciCharCpy(entry->name, terminalName);

    /* Create terminal and save its handle */
    entry->handle = cniCreateInputTerminal(&rc, context->nodeObject, (CciChar*)terminalName);

    /* Link an existing previous element to this one */
    if (context->inputTerminalListPrevious) context->inputTerminalListPrevious->next = entry;
    else if ((context->inputTerminalListHead) == 0) context->inputTerminalListHead = entry;

    /* Save the pointer to the previous element */
    context->inputTerminalListPrevious = entry;
  }
  else {
    /* Error: Unable to allocate memory */
  }

  return(entry);
}
```

   The following example shows the code for `insOutputTerminalListEntry`:

```
TERMINAL_LIST_ENTRY *insOutputTerminalListEntry(
  NODE_CONTEXT_ST* context,
  CciChar*         terminalName
){
  static char* functionName = (char *)"insOutputTerminalListEntry()";
  TERMINAL_LIST_ENTRY* entry;
```

```
  int                 rc;

  entry = (TERMINAL_LIST_ENTRY *)malloc(sizeof(TERMINAL_LIST_ENTRY));
  if (entry) {

    /* This entry is the current end of the list */
    entry->next = 0;

    /* Store the terminal name */
    CciCharCpy(entry->name, terminalName);

    /* Create terminal and save its handle */
    entry->handle = cniCreateOutputTerminal(&rc, context->nodeObject, (CciChar*)terminalName);

    /* Link an existing previous element to this one */
    if (context->outputTerminalListPrevious) context->outputTerminalListPrevious->next = entry;
    else if ((context->outputTerminalListHead) == 0) context->outputTerminalListHead = entry;

    /* Save the pointer to the previous element */
    context->outputTerminalListPrevious = entry;
  }
  else {
    /* Error: Unable to allocate memory */
  }

  return(entry);
}
```

For the minimum code required to compile a C user-defined node, see C skeleton code.

Setting attributes

About this task

Attributes are set whenever you start the integration node, or when you redeploy a message flow with new values. Attributes are set by the integration node calling user code on the configuration thread. Your code needs to store these attributes in its node context area, for later use when processing messages.

Following the creation of input and output terminals, the integration node calls the `cniSetAttribute` function to pass the values for the configured attributes for this instantiation of the user-defined node. For example:

```
    {
      const CciChar* ucsAttr = CciString("nodeTraceSetting", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_INTEGER);
      _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constZero);
      free((void *)ucsAttr) ;
    }
    {
      const CciChar* ucsAttr = CciString("nodeTraceOutfile", BIP_DEF_COMP_CCSID) ;
      insAttrTblEntry(p, (CciChar*)ucsAttr, CNI_TYPE_STRING);
      _setAttribute(p, (CciChar*)ucsAttr, (CciChar*)constSwitchTraceLocation);
      free((void *)ucsAttr) ;
    }
```

The number of configuration attributes that a node can have is unlimited. However, a node must not implement an attribute that is already implemented as a base configuration attribute. The following list shows base attributes:

- label
- userTraceLevel
- traceLevel
- userTraceFilter
- traceFilter

Implementing the node functionality

About this task

When the integration node retrieves a message from the queue, and that message arrives at the input terminal of your user-defined message processing or output node, the integration node calls the implementation function `cniEvaluate`. This function is called on the message processing thread and it must decide what to do with the message. This function might be called on multiple threads, especially if additional instances are used.

Deleting an instance of the node

About this task

If a node is deleted, the integration node calls the `cniDeleteNodeContext` function. This function is started on the same thread as `cniCreateNodeContext`. Use this function to release resources used by your user-defined node. For example:

```
void _deleteNodeContext(
  CciContext* context
){
  static char* functionName = (char *)"_deleteNodeContext()";
  free ((void*) context);
  return;
}
```

Extending the capability of a C message processing or output node
When you have created a user-defined message processing or output node in C, you can extend its capability.

Before you begin

Read the topic "Creating a message processing or output node in C" on page 2392.

About this task

After you have created a user-defined node, the following options are available:

1. "Accessing message data" on page 2397
2. "Transforming a message object" on page 2398
3. "Accessing ESQL" on page 2399
4. "Propagating a message" on page 2399
5. "Writing to an output device" on page 2400

Accessing message data

About this task

In many cases, the user-defined node must access the contents of the message that is received on its input terminal. The message is represented as a tree of syntax elements. Groups of utility functions are provided for message management, message buffer access, syntax element navigation, and syntax element access. (See C node utility functions for details of the utility functions.)

The types of query that you are likely to want to perform include:

- Obtaining the root element of the required message object
- Accessing the bit stream representation of an element tree
- Navigating or querying the tree by asking for child or sibling elements by name
- Getting the type of the element

- Getting the value of the element

For example, to query the name and type of the first child of body:

```
void cniEvaluate( ...
){
  ...
/* Navigate to the target element */
  rootElement = cniRootElement(&rc, message);
  bodyElement = cniLastChild(&rc, rootElement);
  bodyFirstChild = cniFirstChild(&rc, bodyElement);

/* Query the name and value of the target element */
  cniElementName(&rc, bodyFirstChild, (CciChar*)&elementname, sizeof(elementName));
  bytes = cniElementCharacterValue(
  &rc, bodyfirstChild, (CciChar*)&eValue, sizeof(eValue));
  ...
}
```

To access the bit stream representation of an element tree you can use the `cniElementAsBitstream` function. Using this function, you can obtain the bit stream representation of any element in a message. See cniElementAsBitstream for details of how to use this function, and sample code.

Transforming a message object

About this task

The received input message is read-only, therefore before a message can be transformed, you must write it to a new output message using the `cniCreateMessage` function. You can copy elements from the input message, or you can create new elements and attach them to the message. New elements are typically in a parser's domain.

For example:

1. To write the incoming message to a new message:

   ```
   {
     ...
     context = cniGetMessageContext(&rc, message));
     outMsg = cniCreateMessage(&rc, context));
     ...
   }
   ```

2. To make a copy of the new message:

   ```
   cniCopyElementTree(&rc, sourceElement, targetElement);
   ```

3. To modify the value of a target element:

   ```
     cniSetElementIntegerValue(&rc, targetElement, L"newValue", 8);
   ```

4. After finalizing and propagating the message, you must delete the output message using the `cniDeleteMessage` function:

   ```
    cniDeleteMessage(&rc, outMsg);
   ```

As part of the transformation, you might want to create a new message body. To create a new message body, use one of the following functions, which assign a parser to a message tree folder:

```
cniCreateElementAsFirstChildUsingParser
cniCreateElementAsLastChildUsingParser
cniCreateElementAfterUsingParser
cniCreateElementBeforeUsingParser
```

When creating a message body, do not use the following functions because they do not associate an owning parser with the folder:

```
cniCreateElementAsFirstChild
cniCreateElementAsLastChild
```

```
cniCreateElementAfter
cniCreateElementBefore
```

Accessing ESQL

About this task

Nodes can invoke ESQL expressions using Compute node ESQL syntax. You can create and modify the components of the message using ESQL expressions, and you can refer to elements of both the input message and data from an external database using the `cniSqlCreateStatement`, `cniSqlSelect`, `cniSqlDeleteStatement`, and `cniSqlExecute` functions.

For example, to populate the Result element from the contents of a column in a database table:

```
{
  ...
  sqlExpr = cniSqlCreateStatement(&rc,
   (NODE_CONTEXT_ST *)context->nodeObject,
   L"DB", CCI_SQL_TRANSACTION_AUTO,
   L"SET OutputRoot.XMLNS.Result[] = (SELECT T.C1 AS Col1 FROM Database.TABLE AS T;");
  ...
  cniSqlSelect(&rc, sqlExpr, localEnvironment, exceptionList, message, outMsg);
  cniSqlDeleteStatement(&rc, sqlExpr);
  ...
}
```

For more information about ESQL, see “ESQL overview” on page 1608.

If your user-defined node primarily uses ESQL, consider using a node.

Propagating a message

About this task

Before you propagate a message, decide what message flow data you want to propagate, and which terminal is to receive the data.

1. If the message has changed, finalize the message before you propagate it using the `cniFinalize` function. For example:

   ```
         cniFinalize(&rc, outMsg, CCI_FINALIZE_NONE);
   ```

2. The terminalObject is derived from a list that the user-defined node maintains itself. To propagate the message to the output terminal, use the `cniPropagate` function:

   ```
   if (terminalObject) {
     if (cniIsTerminalAttached(&rc, terminalObject)) {
       if (rc == CCI_SUCCESS) {
         cniPropagate(&rc, terminalObject, localEnvironment, exceptionList, outMsg);
       }
     }
   ```

   In the above example, the `cniIsTerminalAttached` function is used to test whether the message can be propagated to the specified terminal. If you do not use the `cniIsTerminalAttached` function and the terminal is not attached to another node by a connector, the message is not propagated and no warning message is returned. Use the `cniIsTerminalAttached` function to prevent this error occurring.

3. If you created a new output message using `cniCreateMessage`, after propagating the message, delete the output message using the `cniDeleteMessage` function:

   ```
   cniDeleteMessage(&rc, outMsg);
   ```

Writing to an output device

About this task

A transformed message must be serialized to a bit stream; a message can be serialized only once.

The bit stream can then be accessed and written to an output device. Write the message to a bit stream using the `cniWriteBuffer` function. For example:

```
{
  ...
  cniWriteBuffer(&rc, message);
  writeToDevice(cniBufferPointer(&rc, message), cniBufferSize(&rc, message));
  ...
}
```

In this example, the method `writeToDevice` is a user-written method which writes a bit stream to an output device.

Do not write a user-defined output node to write messages to IBM MQ queues; use the supplied MQOutput node in this scenario. The integration node internally maintains an IBM MQ connection and open queue handles on a thread-by-thread basis, and these are cached to optimize performance. In addition, the integration node handles recovery scenarios when certain IBM MQ events occur; this recovery would be adversely affected if IBM MQ MQI calls are used in a user-defined output node.

Compiling a C user-defined extension
Compile user-defined extensions in C for all supported operating systems.

Before you begin

If you create your own user-defined nodes, parsers, and user exits in C, compile them on the operating system on which the target integration node is running. Samples are provided for both nodes and parsers, and are described in Sample node files and Sample parser files. Use the instructions here to compile the samples. If you want to create your own extensions, see the following topics:

- “Creating a user-defined extension in C” on page 2384
- “Developing user-defined parsers” on page 2434
- “Developing user-defined exits” on page 2443

About this task

These instructions use the file names of the supplied samples. If you are compiling your own user-defined extensions, substitute your own file names.

When you compile a user-defined extension that is written in C, you need a compatible compiler. For details of supported compilers, see IBM App Connect Enterprise system requirements.

Header files

About this task

The following header files define the C interfaces:

BipCni.h
Message processing nodes

BipCpi.h
Message parsers

BipCci.h
Interfaces common to both nodes and parsers

BipCos.h
Platform-specific definitions

Compiling

About this task

Compile the source for your user-defined extension on each of the supported operating systems to create the executable file that the integration node calls to implement your user-defined extension. On Linux and AIX and systems, this file is a loadable implementation library (LIL) file; on Windows systems, it is a dynamic load library (DLL) file.

The libraries that you build to contain user-defined nodes or parsers must have the extension `.lil` on all operating systems so that the integration node can load them. Libraries that contain user exits must have the extension `.lel` on all operating systems. The examples in this topic show libraries with the extension `.lil`

All extensions must be compiled as 64-bit libraries.

Refer to the documentation for the compiler that you are using for full details of available compile and link options that might be required for your programs.

Navigate to the directory where your user-defined extension source code is located, and follow the instructions for your operating system:

- AIX
- Linux
- Windows

Compiling on AIX

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension:

```
xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c SwitchNode.c \
   -o SwitchNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c TransformNode.c \
   -o TransformNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c BipSampPluginUtil.c \
   -o BipSampPluginUtil.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c Common.c \
   -o Common.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c NodeFactory.c \
   -o NodeFactory.o

xlc_r -q64 \
      -qmkshrobj \
      -bM:SRE \
      -bexpall \
      -bnoentry \
      -o SwitchNode.lil SwitchNode.o \
         BipSampPluginUtil.o Common.o NodeFactory.o \
```

```
      -L /install_dir/server/lib \
      -l imbdfplg

chmod a+r SwitchNode.lil
```

Compiling on Linux

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

When you compile programs on Linux on POWER, replace the option `-fpic` with `-fPIC` if you want to use dynamic linking and avoid any limit on the size of the global offset table.

The following instructions are for compiling an extension on Linux on x86-64, Linux on POWER, and Linux on Z. To compile the extension for Linux on POWER, include the **-02** parameter in the compile and link examples, and include the **-fexceptions** parameter in the compile examples.

```
g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   TransformNode.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   SwitchNode.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   BipSampPluginUtil.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX  -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   Common.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   NodeFactory.c

g++ -m64 -o samples.lil \
   TransformNode.o \
   SwitchNode.o \
   BipSampPluginUtil.o \
   Common.o NodeFactory.o \
   -shared -lc -lnsl -ldl \
   -L/install_dir/server/lib -limbdfplg
```

These commands create the file `samples.lil` that provides TransformNode and SwitchNode objects.

Compiling on Windows

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Ensure that you include a space between `SwitchNode.c` and `BipSampPluginUtil.c`, and also between `-link` and `/DLL`.

Enter the command as a single line of input; in the following example the lines are split to improve readability.

```
cl /VERBOSE /LD /MD /Zi /EHsc /I.
   /Iinstall_dir\server\include\plugin
   SwitchNode.c BipSampPluginUtil.c Common.c
   NodeFactory.c TransformNode.c
   -link /DLL install_dir\server\lib\imbdfplg.lib
   /OUT:SwitchNode.lil
```

If you have correctly set the *LIB* environment variable, you do not have to specify the full paths to the LIB files.

Creating a user-defined extension in Java
You must complete a series of tasks to create user-defined nodes that use the Java language.

About this task

Complete the appropriate tasks from the following list:

- “Creating an input node in Java” on page 2403
- “Creating a message processing or output node in Java” on page 2409
- “Compiling a Java user-defined node” on page 2418
- “Packaging a Java user-defined node ” on page 2450

You can write only user-defined nodes in Java: you must write user-defined parsers in C.

What to do next

When you have completed this set of tasks, continue with the following tasks:

- “Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit” on page 2421
- “Testing a user-defined node” on page 2430
- “Packaging and distributing user-defined extensions” on page 2449

Restrictions when creating Java nodes
In Java user-defined nodes and the JavaCompute node, calling the `System.exit(...)` method is not supported. Calling this method results in a `SecurityException`.

Creating an input node in Java
An input node is used to receive a message into a message flow, typically from a source that is not supported by the built-in input nodes.

Before you begin

Read the following topics:

- “Why use a user-defined extension?” on page 2322
- “User-defined input nodes” on page 2332

About this task

To create an input node in the Java language:

1. “Creating a Java project” on page 2404
2. “Declaring the input node class” on page 2404
3. “Defining the node constructor” on page 2405
4. “Receiving external data into a buffer” on page 2405
5. “Propagating the message” on page 2406

6. “Controlling threading and transactionality” on page 2406
7. “Declaring the node name” on page 2407
8. “Declaring attributes” on page 2407
9. “Implementing the node functionality” on page 2408
10. “Overriding default message parser attributes (optional)” on page 2408
11. “Deleting an instance of the node” on page 2409

A Java user-defined node is distributed as a `.jar` file.

Creating a Java project

About this task

Before you can create Java nodes in the IBM App Connect Enterprise Toolkit, you must create a Java project:

Procedure

1. Click **File** > **New** > **Project**, select **Java Project** then click **Next**.
2. Give the project a name, then click **Next**.
3. On the **Java Settings** pane, select the **Libraries** tab, and click **Add External JARs**.
4. Select *install_dir*`\server\classes\jplugin2.jar`, where *install_dir* is the home directory of your IBM App Connect Enterprise installation.
5. Follow the prompts on the other tabs to define any other build settings.
6. Click **Finish**.

What to do next

You can now develop the source for your Java node in this project.

Declaring the input node class

About this task

Every class that implements `MbInputNodeInterface` and is contained in the integration node LIL path is registered with the integration node as an input node. When you implement `MbInputNodeInterface`, you must also implement a `run` method for this class. The `run` method represents the start of the message flow, contains the data that formulates the message, and propagates it down the flow. The integration node calls the `run` method when threads become available in accordance with your specified threading model.

The class name must end with the word "Node". For example, if the name is BasicInput in the IBM App Connect Enterprise Toolkit, the class name must be BasicInputNode.

For example, to declare the input node class:

```
package com.ibm.jplugins;

import com.ibm.broker.plugin.*;

public class BasicInputNode extends MbInputNode implements MbInputNodeInterface
{
...
```

Follow these steps to complete this action in the IBM App Connect Enterprise Toolkit:

Procedure

1. Click **File** > **New** > **Other**, select **Class**, then click **Next**.

2. Set the package and class name fields to appropriate values.
3. Delete the text in the **Superclass** text field and click the **Browse** button.
4. Select **MbInputNode**.
5. Click the **Add** button next to Interfaces text field, and select **MbInputNodeInterface**.
6. Click **Finish**.

Defining the node constructor

About this task

When the node is instantiated, the constructor of the node class is called. This class is where you create the terminals of the node, and initialize the default values for the attributes.

An input node has a number of output terminals associated with it, but does not typically have any input terminals. Use the `createOutputTerminal` method to add output terminals to a node when the node is instantiated. For example, to create a node with three output terminals:

```
public BasicInputNode() throws MbException
{
 createOutputTerminal ("out");
 createOutputTerminal ("failure");
 createOutputTerminal ("catch");
 setAttribute ("firstParserClassName","myParser");
 attributeVariable  = "none";
}
```

Receiving external data into a buffer

About this task

An input node can receive data from any type of external source, such as a file system, a queue, or a database, in the same way as all other Java programs, if the output from the node is in the correct format.

Provide an input buffer (or bit stream) to contain input data, and associate it with a message object. Create a message from a byte array by using the `createMessage` method of the **MbInputNode** class, and then generate a valid message assembly from this message. For example, to read the input data from a file:

Procedure

1. Create an input stream to read from the file:

   ```
   FileInputStream inputStream = new FileInputStream("myfile.msg");
   ```

2. Create a byte array the size of the input file:

   ```
   byte[] buffer = new byte[inputStream.available()];
   ```

3. Read from the file into the byte array:

   ```
   inputStream.read(buffer);
   ```

4. Close the input stream:

   ```
   inputStream.close();
   ```

5. Create a message to put on the queue:

   ```
   MbMessage msg = createMessage(buffer);
   ```

6. Create a message assembly to hold this message:

   ```
   msg.finalizeMessage(MbMessage.FINALIZE_VALIDATE);
   MbMessageAssembly newAssembly =
   ```

```
      new MbMessageAssembly(assembly, msg);
```

Propagating the message

About this task

After creating a message assembly, you can propagate it to one of the output terminals that are defined on the node.

For example, to propagate the message assembly to the terminal named **out**:

```
MbOutputTerminal out = getOutputTerminal("out");
out.propagate(newAssembly);
```

To delete the message:

```
msg.clearMessage();
```

To clear the memory that is allocated for the message tree, call the `clearMessage()` function within the finally block of `try/catch`.

Controlling threading and transactionality

About this task

The integration node infrastructure handles transaction issues such as controlling the commit of an IBM MQ or database unit of work when message processing has completed. However, resources modified from within a user-defined node are not necessarily under the transactional control of the integration node.

Each message flow thread is allocated from a pool of threads maintained for each message flow, and starts in the `run` method.

The user-defined node uses return values to indicate whether a transaction is successful, to control whether transactions are committed or rolled back, and to control when the thread is returned to the pool. The integration node infrastructure catches all unhandled exceptions, and rolls back the transaction.

You determine the behavior of transactions and threads by using the appropriate return value:

MbInputNode.SUCCESS_CONTINUE
: The transaction is committed and the integration node calls the `run` method again by using the same thread.

MbInputNode.SUCCESS_RETURN
: The transaction is committed and the thread is returned to the thread pool, assuming that it is not the only thread for this message flow.

MbInputNode.FAILURE_CONTINUE
: The transaction is rolled back and the integration node calls the `run` method again by using the same thread.

MbInputNode.FAILURE_RETURN
: The transaction is rolled back and the thread is returned to the thread pool, assuming that it is not the only thread for this message flow.

MbInputNode.TIMEOUT
: The `run` method must not block indefinitely while waiting for input data to arrive. While the flow is blocked by user code, you cannot shut down or reconfigure the integration node. The `run` method must yield control to the integration node periodically by returning from the `run` method. If input data is not received after a certain period (for example, 5 seconds), the method must return with the TIMEOUT return code. Assuming that the integration node does not need to reconfigure or shut down, the `run` method of the input node gets called again immediately.

To create multithreaded message flows, you call the `dispatchThread` method after a message is created, but before the message is propagated to an output terminal. This action ensures that only one thread is waiting for data while other threads are processing the message. New threads are obtained

from the thread pool up to the maximum limit specified by the `Additional Instances` property of the message flow. For example:

```
public int run( MbMessageAssembly assembly ) throws MbException
{
  byte[] data = getDataWithTimeout();  // user supplied method
                                       // returns null if timeout
  if( data == null )
    return TIMEOUT;

  MbMessage msg = createMessage( data );
  msg.finalizeMessage( MbMessage.FINALIZE_VALIDATE );
  MbMessageAssembly newAssembly =
       new MbMessageAssembly( assembly, msg );

  dispatchThread();

  getOutputTerminal( "out" ).propagate( newAssembly );

  return SUCCESS_RETURN;
}
```

Declaring the node name

About this task

You must declare the name of the node for use and identification by the IBM App Connect Enterprise Toolkit. All node names must end with the characters "Node". Declare the name by using the following method:

```
public static String getNodeName()
{
   return "BasicInputNode";
}
```

If this method is not declared, the Java API framework creates a default node name by using the following rules:

- The class name is appended to the package name.
- The periods are removed, and the first letter of each part of the package and class name is capitalized.

For example, by default, the following class is assigned the node name "ComIbmPluginsamplesBasicInputNode":

```
package com.ibm.pluginsamples;
public class BasicInputNode extends MbInputNode implements MbInputNodeInterface
{
   ...
```

Declaring attributes

About this task

Declare node attributes by using the same method that you use for Java bean properties. You are responsible for writing `get` and `set` methods for the attributes; the API framework infers the attribute names by using the Java bean introspection rules. For example, if you declare the following two methods:

```
private String attributeVariable;

public String getFirstAttribute()
{
  return attributeVariable;
}

public void setFirstAttribute(String value)
{
```

```
  attributeVariable = value;
}
```

The integration node infers that this node has an attribute called firstAttribute. This name is derived from the names of the `get` or `set` methods, not from the variable names of any internal class members. Attributes can be exposed only as strings, so convert numeric types to and from strings in the `get` or `set` methods. For example, the following method defines an attribute called timeInSeconds:

```
int seconds;

public String getTimeInSeconds()
{
  return Integer.toString(seconds);
}

public void setTimeInSeconds(String value)
{
  seconds = Integer.parseInt(value);
}
```

Implementing the node functionality

About this task

As already described, the `run` method is called by the integration node to create the input message. This method must provide all the processing function for the input node.

Overriding default message parser attributes (optional)

About this task

An input node implementation normally determines which message parser initially parses an input message. For example, the built-in MQInput node dictates that an MQMD parser is required to parse the MQMD header. A user-defined input node can select an appropriate header or message parser, and the mode in which the parsing is controlled, by using the following default attributes that are included, which you can override:

rootParserClassName
: Defines the name of the root parser that parses message formats supported by the user-defined input node. It defaults to `GenericRoot`, a supplied root parser that causes the integration node to allocate and chain parsers together. It is unlikely that a node would have to modify this attribute value.

firstParserClassName
: Defines the name of the first parser, in what might be a chain of parsers that are responsible for parsing the bit stream. It defaults to `XML`.

messageDomainProperty
: An optional attribute that defines the name of the message parser required to parse the input message. The supported values are the same as the values that are supported by the MQInput node.

messageSetProperty
: An optional attribute that defines the message set identifier, or the message set name, in the `Message Set` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageTypeProperty
: An optional attribute that defines the identifier of the message in the `MessageType` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

messageFormatProperty
: An optional attribute that defines the format of the message in the `Message Format` field, only if the MRM parser was specified by the `messageDomainProperty` attribute.

Deleting an instance of the node

About this task

An instance of the node is deleted when either:

- You shut down the integration node.
- You remove the node or the message flow that contains the node, and redeploy the configuration.

When the node is deleted, it can perform cleanup operations, such as closing sockets, if it implements the optional `onDelete` method. This method, if present, is called by the integration node just before the node is deleted.

Implement the `onDelete` method as follows:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

Creating a message processing or output node in Java
A message processing node is used to process a message, and an output node is used to output a message as a bit stream.

Before you begin

Read the following topics:

- “Why use a user-defined extension?” on page 2322
- “User-defined message processing nodes” on page 2336
- “User-defined output nodes” on page 2343
- “Restrictions when creating Java nodes” on page 2403

About this task

IBM App Connect Enterprise provides the source for two sample user-defined nodes called SwitchNode and TransformNode. You can use these nodes in their current state, or you can modify them.

When you code a message processing node or an output node, the two types provide essentially the same functions. You can perform message processing within an output node, and likewise you can propagate an output message to a bit stream from a message processing node. For simplicity, this topic refers mainly to the node as a message processing node, but it does discuss the functionality of both types of node.

Complete the following steps:

1. “Creating a new Java project” on page 2410
2. “Declaring the message processing node class” on page 2410
3. “Defining the node constructor” on page 2410
4. “Accessing message data” on page 2411
5. “Transforming a message object” on page 2411
6. “Propagating the message” on page 2412
7. “Declaring the node name” on page 2413
8. “Declaring attributes” on page 2413
9. “Implementing the node functionality” on page 2414
10. “Deleting an instance of the node” on page 2414

A Java user-defined node is distributed as a `.jar` file.

Creating a new Java project

About this task

Before you can create Java nodes in the IBM App Connect Enterprise Toolkit, you must create a new Java project:

1. Click **File** > **New** > **Project**. Select **Java** and click **Next**.
2. In the **Project name** field, enter a project a name, then click **Next**.
3. On the **Java Settings** pane, select the **Libraries** tab, and click **Add External JARs**.
4. Select *install_dir*`\server\classes\jplugin2.jar`.
5. Follow the prompts on the other tabs to define any other build settings.
6. Click **Finish**.

You can now develop the source for your Java node within this project.

Declaring the message processing node class

About this task

Any class that implements `MbNodeInterface`, and is contained in the integration node's LIL path, is registered with the integration node as a message processing node. When you implement `MbNodeInterface`, you must also implement an `evaluate` method for this class. The `evaluate` method is called by the integration node for each message that passes through the flow.

The class name must end with the word "Node". For example, if you have assigned the name as Basic in the IBM App Connect Enterprise Toolkit, the class name must be BasicNode.

For example, to declare the message processing node class:

```
package com.ibm.jplugins;

import com.ibm.broker.plugin.*;

public class BasicNode extends MbNode implements MbNodeInterface
```

Declare the class in the IBM App Connect Enterprise Toolkit:

1. Click **File** > **New** > **Other**, select **Class**, then click **Next**.
2. Set the package and class name fields to appropriate values.
3. Delete the text in the Superclass text field and click **Browse**.
4. Select **MbNode** and click **OK**.
5. Click the **Add** button next to Interfaces text field, and select **MbNodeInterface**.
6. Click **Finish**.

Defining the node constructor

About this task

When the node is instantiated, the constructor of the user's node class is called. Create the terminals of the node, and initialize any default values for attributes in this constructor.

A message processing node has a number of input terminals and output terminals that are associated with it. Use the methods `createInputTerminal` and `createOutputTerminal` to add terminals to a node when the node is instantiated.

For example, to create a node with one input terminal and two output terminals:

```
public MyNode() throws MbException
{
  // create terminals here
```

```
  createInputTerminal ("in");
  createOutputTerminal ("out");
  createOutputTerminal ("failure");
}
```

Accessing message data

About this task

In many cases, the user-defined node needs to access the contents of the message received on its input terminal. The message is represented as a tree of syntax elements. Use the supplied utility function to `evaluate` methods for message management, message buffer access, syntax element navigation, and syntax element access.

The `MbElement` class provides the interface to the syntax elements.

For example:

Procedure

1. To navigate to the relevant syntax element in the XML message:

```
MbElement rootElement = assembly.getMessage().getRootElement();
MbElement switchElement =
rootElement.getLastChild().getFirstChild().getFirstChild();
```

2. To select the terminal indicated by the value of this element:

```
String terminalName;
String elementValue = (String)switchElement.getValue();
if(elementValue.equals("add"))
  terminalName = "add";
else if(elementValue.equals("change"))
  terminalName = "change";
else if(elementValue.equals("delete"))
  terminalName = "delete";
else if(elementValue.equals("hold"))
  terminalName = "hold";
else
  terminalName = "failure";

MbOutputTerminal out = getOutputTerminal(terminalName);
```

Results

Transforming a message object

About this task

The received input message is read-only, so before you can transform a message, you must write it to a new output message. You can copy elements from the input message, or you can create new elements in the output message.

The `MbMessage` class provides the copy constructors, and the methods to get the root element of the message. The `MbElement` class provides the interface to the syntax elements.

For example, if you have an incoming message assembly with embedded messages, you could have the following code in the `evaluate` method of your user-defined node:

1. To create a new copy of the message assembly and its embedded messages:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);
```

2. To navigate to the relevant syntax element in the XML message:

```
MbElement rootElement = newAssembly.getMessage().getRootElement();
```

```
    MbElement switchElement =
   rootElement.getFirstElementByPath("/XML/data/action");
```

3. To change the value of an existing element:

```
  String elementValue = (String)switchElement.getValue();
    if(elementValue.equals("add"))
      switchElement.setValue("change");
    else if(elementValue.equals("change"))
      switchElement.setValue("delete");
    else if(elementValue.equals("delete"))
      switchElement.setValue("hold");
    else
      switchElement.setValue("failure");
```

4. To add a new tag as a child of the switch tag:

```
    MbElement tag = switchElement.createElementAsLastChild(MbElement.TYPE_NAME,
                                                           "PreviousValue",
                                                           elementValue);
```

5. To add an attribute to this new tag:

```
    tag.createElementAsFirstChild(MbElement.TYPE_NAME_VALUE,
                                  "NewValue",
                                  switchElement.getValue());

    MbOutputTerminal out = getOutputTerminal("out");
```

As part of the transformation, you might need to create a new message body. To create a new message body, use one of the following methods, which specifically assigns a parser to a message tree folder:

```
createElementAfter(String)
createElementAsFirstChild(String)
createElementAsLastChild(String)
createElementBefore(String)
createElementAsLastChildFromBitstream(byte[], String, String, String, String, int, int, int)
```

Do not use the following methods, which do not associate an owning parser with the folder:

```
createElementAfter(int)
createElementAfter(int, String, Object)
createElementAsFirstChild(int)
createElementAsFirstChild(int, String, Object)
createElementAsLastChild(int)
createElementAsLastChild(int, String, Object)
createElementBefore(int)
createElementBefore(int, String, Object)
```

Propagating the message

About this task

Before you propagate a message, decide what message flow data you want to propagate, and whether to propagate to a node terminal, or to a Label node.

For example:

1. To propagate the message to the output terminal "out":

```
MbOutputTerminal out = getOutputTerminal("out");
        out.propagate(newAssembly);
```

2. To propagate the message to a Label node:

```
MbRoute label1 = getRoute ("label1");
Label1.propagate(newAssembly);
```

Call the `clearMessage()` function within the finally block of try/catch to clear the memory that is allocated for the message tree.

To propagate the same MbMessage object multiple times, call the `finalizeMessage()` method on the MBMessage object, so that any changes made to the message are reflected in the bit stream that is generated downstream of the Java node; for example:

```
MbMessage newMsg = new MbMessage(assembly.getMessage());
MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(newAssembly;
...
newMsg.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(newAssembly);
```

Declaring the node name

About this task

The name of the node must be the same as the one that is used in the IBM App Connect Enterprise Toolkit. All node names must end with "Node". Declare the name using the following method:

```
public static String getNodeName()
{
   return "BasicNode";
}
```

If this method is not declared, the Java API framework creates a default node name using the following rules:

- The class name is appended to the package name.
- The dots are removed, and the first letter of each part of the package and class name are capitalized.

For example, by default, the following class is assigned the node name "ComIbmPluginsamplesBasicNode":

```
package com.ibm.pluginsamples;
public class BasicNode extends MbNode implements MbNodeInterface
{
   ...
```

Declaring attributes

About this task

Declare node attributes in the same way as Java Bean properties. You must write getter and setter methods for the attributes. The API framework infers the attribute names using the Java Bean introspection rules. For example, if you declare the following two methods:

```
private String attributeVariable;

public String getFirstAttribute()
{
  return attributeVariable;
}

public void setFirstAttribute(String value)
{
  attributeVariable = value;
}
```

the integration node infers that this node has an attribute called firstAttribute. This name is derived from the names of the get or set methods, not from any internal class member variable names. Attributes can

only be exposed as strings, therefore, you must convert any numeric types to and from strings in the get or set methods. For example, the following method defines an attribute called timeInSeconds:

```
int seconds;

public String getTimeInSeconds()
{
  return Integer.toString(seconds);
}

public void setTimeInSeconds(String value)
{
  seconds = Integer.parseInt(value);
}
```

Implementing the node functionality

About this task

The `evaluate` method, defined in `MbNodeInterface`, is called by the integration node to process the message. All the processing function for the node is included in this method.

The `evaluate` method has two parameters that are passed in by the integration node:

1. The `MbMessageAssembly`, which contains the following objects that are accessed using the appropriate methods:
   - The incoming message
   - The LocalEnvironment
   - The global Environment
   - The ExceptionList
2. The input terminal on which the message has arrived.

For example, the following code extract shows how you might write the evaluate method:

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    // add message processing code here

    getOutputTerminal("out").propagate(assembly);
  }
```

The message flow data, which consists of the message, Environment, LocalEnvironment, and ExceptionList, is received at the input terminal of the node.

Deleting an instance of the node

About this task

An instance of the node is deleted when either:

- You shut down the integration node.
- You remove the node or the message flow that contains the node, and redeploy the configuration.

If you want the node to perform any cleanup operations, for example closing sockets, include an implementation of the `onDelete` method:

```
public void onDelete()
{
  // perform node cleanup if necessary
}
```

This method is called by the integration node immediately before it deletes the node.

Extending the capability of a Java message processing or output node
Within a message processing or output node, you can add extended functions to your Java node.

Before you begin

Read "Creating a message processing or output node in Java" on page 2409.

About this task

You can add one or more of the following functions:

- "Accessing ESQL" on page 2415
- "Interacting with databases" on page 2416
- "Handling exceptions" on page 2416
- "Writing to an output device" on page 2417

Accessing ESQL

About this task

Nodes can invoke ESQL expressions using Compute node ESQL syntax. You can create and modify the components of the message using ESQL expressions, and you can refer to elements of both the input message and data from an external database.

The following procedure demonstrates how to use ESQL to control transactions from the `evaluate` method in your user-defined node:

1. Set the name of the ODBC data source to use. For example:

```
String dataSourceName = "myDataSource";
```

2. Set the ESQL statement to run:

```
String statement =
   "SET OutputRoot.XMLNS.data =
          (SELECT Field2 FROM Database.Table1 WHERE Field1 = 1);";
```

   Or, if you want to run a statement that returns no result:

```
String statement = "PASSTHRU(
                        'INSERT INTO Database.Table1 VALUES(
                             InputRoot.XMLNS.DataField1,
                             InputRoot.XMLNS.DataField2)');";
```

3. Select the transaction you want from the following types:

   MbSQLStatement.SQL_TRANSACTION_COMMIT
   Immediately commit the transaction after the ESQL statement has completed.

   MbSQLStatement.SQL_TRANSACTION_AUTO
   Commit the transaction when the message flow has completed. (Rollbacks are performed if necessary.)

   For example:

```
int transactionType = MbSQLStatement.SQL_TRANSACTION_AUTO;
```

4. Get the ESQL statement. For example:

```
MbSQLStatement sql =
       createSQLStatement(dataSourceName, statement, transactionType);
```

   You can use the method `createSQLStatement(data source, statement` to default the transaction type to MbSQLStatement.SQL_TRANSACTION_AUTO).

5. Create the new message assembly to be propagated:

```
MbMessageAssembly newAssembly =
       new MbMessageAssembly(assembly, assembly.getMessage());
```

6. Run the ESQL statement:

```
sql.select(assembly, newAssembly);
```

   Or, if you want to run an ESQL statement that returns no result:

```
sql.execute(assembly);
```

Interacting with databases

About this task

You can interact with databases from the Java code in your message processing node. The support that is provided is identical to the support for Java code that you write for the JavaCompute node; for details of the available options, and the advantages and restrictions that apply, see “Interacting with databases by using the JavaCompute node” on page 1785.

Handling exceptions

About this task

Use the `MbException` class to catch and access exceptions. The `MbException` class returns an array of exception objects that represent the children of an exception in the integration node exception list. Each element returned specifies its exception type. An empty array is returned if an exception has no children. The following code sample shows an example of how you might use the `MbException` class in the `evaluate` method of your user-defined node.

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    try
      {

        // plug-in functionality

      }
    catch(MbException ex)
      {
        traverse(ex, 0);

        throw ex; // if re-throwing, it must be the original exception that was caught
      }
  }

  void traverse(MbException ex, int level)
  {
    if(ex != null)
      {
        // Do whatever action here
        System.out.println("Level: " + level);
        System.out.println(ex.toString());
        System.out.println("traceText:  " + ex.getTraceText());

        // traverse the hierarchy
        MbException e[] = ex.getNestedExceptions();
        int size = e.length;
        for(int i = 0; i < size; i++)
          {
            traverse(e[i], level + 1);
          }
      }
  }
```

You can develop a user-defined message-processing or output node in such a way that it can access all current exceptions. For example, to catch database exceptions you can use the `MbSQLStatement` class.

This class sets the value of the 'throwExceptionOnDatabaseError' attribute, which determines integration node behavior when it encounters a database error. When it is set to true, if an exception is thrown it can be caught and handled by the `evaluate` method in your user-defined extension.

The following code sample shows an example of how to use the `MbSQLStatement` class.

```
public void evaluate(MbMessageAssembly assembly, MbInputTerminal inTerm) throws MbException
  {
    MbMessage newMsg = new MbMessage(assembly.getMessage());
    MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

    String table =
       assembly.getMessage().getRootElement().getLastChild().getFirstChild().getName();

    MbSQLStatement state = createSQLStatement( "dbName",
       "SET OutputRoot.XMLNS.integer[] = PASSTHRU('SELECT * FROM " + table + "');" );

    state.setThrowExceptionOnDatabaseError(false);
    state.setTreatWarningsAsErrors(true);

    state.select( assembly, newAssembly );

    int sqlCode = state.getSQLCode();
    if(sqlCode != 0)
      {
        // Do error handling here

        System.out.println("sqlCode = " + sqlCode);
        System.out.println("sqlNativeError = " + state.getSQLNativeError());
        System.out.println("sqlState = " + state.getSQLState());
        System.out.println("sqlErrorText = " + state.getSQLErrorText());
      }

    getOutputTerminal("out").propagate(newAssembly);
  }
```

Writing to an output device

About this task

To write to an output device, the logical (hierarchical) message must be converted back into a bit stream in your `evaluate` method. Use the `getBuffer` method in **MbMessage** to perform this task:

```
public void evaluate( MbMessageAssembly assembly, MbInputTerminal in)
                                                        throws MbException
{
  MbMessage msg = assembly.getMessage();
  byte[] bitstream = msg.getBuffer();

  // write the bitstream out somewhere
  writeBitstream( bitstream );   // user method

 }
```

Typically, for an output node the message is not propagated to any output terminal, therefore you can just return at this point.

You must use the supplied MQOutput node when writing to IBM MQ queues, because the integration node internally maintains an IBM MQ connection and the open queue handles on a thread-by-thread basis. These handles are cached to optimize performance. In addition, the integration node handles exception scenarios when certain IBM MQ events occur, and this recovery is adversely affected if IBM MQ MQI calls are used in a user-defined output node.

Getting and setting the specific type of an Mb element
Two methods are provided for handling the specific type of an Mb syntax element.

About this task

The following methods are available:

- `getSpecificType`
- `setSpecificType`

Use these methods to access or set the specific type of an XML element. For example, to update the current value:

Procedure

1. Call `getSpecificType` on the syntax element.

   The `getSpecificType` method does not take any parameters, but returns the specific type of the element as an int value.
2. Call `setSpecificType` on the syntax element.

   The `setSpecificType` method takes one parameter of the type int, which is the specific type that you want the Mb element to be. This method has no return value.

Results

Specific type values for the XML and MRM parsers are listed in XML, MRM, and XMLNSC parser constants.

Compiling a Java user-defined node

When you have created the code for your Java user-defined node, you must compile it for your operating system.

Before you begin

You must have a user-defined node written in Java. This node can be one of the provided sample nodes that are described in Sample node files, or a node that you have created yourself by using the instructions in either “Creating a message processing or output node in Java” on page 2409 or “Creating an input node in Java” on page 2403.

About this task

You can compile a Java user-defined node either from the command line, or from within the project in the IBM App Connect Enterprise Toolkit. Both options are described later in this section.

When you compile a Java user-defined node from the command line, you must have a compatible IBM Software Developer Kit for Java on the current operating system. For details of supported Java versions, see the IBM App Connect Enterprise system requirements; see IBM App Connect Enterprise system requirements.

Compiling a Java user-defined node from the IBM App Connect Enterprise Toolkit

About this task

Use the following procedure to compile your Java user-defined node from the IBM App Connect Enterprise Toolkit:

Procedure

1. Switch to the Java perspective, and in the Package Explorer, select the `/src` directory inside your node project, and click **File** > **Export**.
2. From the list displayed, select **JAR file**. Click **Next**.

   The resources that are available for you to export as a JAR file are listed.
3. Verify that **Export generated class files and resources** is selected.

4. Specify a name and location for your JAR file.

   Place the file inside the root directory of your node project, and give the file the same name as the name of the project (with a `.jar` extension). You can use the default values for the rest of the options. Click **Finish**.

Results

The `.jar` file that you have created is displayed in your node project, and is ready for you to install on one or more integration nodes (see "Installing user-defined extension runtime files on an integration node" on page 2455), or to package for distribution (see "Packaging and distributing a user-defined node project" on page 2452).

Compiling a Java user-defined node from the command line

About this task

Use the following procedure to compile your Java user-defined node from the command line:

Procedure

1. Add the location of `jplugin2.jar` to the CLASSPATH for your current platform:

   Windows *install_dir*`\server\classes\jplugin2.jar`

   Linux UNIX z/OS *install_dir*`/server/classes/jplugin2.jar`
2. Put your Java user-defined node class into the following directory:

   Windows *install_dir*`\server\sample\extensions\nodes`

   Linux UNIX z/OS *install_dir*`/server/sample/extensions/nodes`
3. Change to the directory that now contains your user-defined node class.
4. Compile the `.java` file by using the **javac** command; for example:

   ```
   javac nodename.java
   ```
5. Package the resulting `.class` file into a `.par` file. See "Packaging a Java user-defined node " on page 2450.

Results

The `.par` file that you have created is ready for you to install on one or more integration nodes (see "Installing user-defined extension runtime files on an integration node" on page 2455), or to package for distribution (see "Packaging and distributing a user-defined node project" on page 2452).

Creating a user-defined node from a subflow
Create user-defined nodes either from scratch, or by using an existing subflow.

About this task

Complete the appropriate tasks from the following list:

- "Creating a user-defined node from a subflow from scratch" on page 2420
- "Creating a user-defined node from an existing subflow" on page 2420
- "Choosing the location of a user-defined node in the palette" on page 2430

Video: Creating and using a subflow user-defined node

This video demonstrates the capability to create a subflow user-defined node using the toolkit. After creating the subflow user-defined node, it is used in another installation of App Connect Enterprise in a message flow to deploy and test the flow.

Creating a user-defined node from a subflow from scratch
Create a user-defined node that packages a subflow by creating a user-defined node project, then creating the user-defined node.

About this task

After you have created a user-defined node, you can edit its properties in the **Palette** editor.
You can define as many user-defined nodes as you want in the same user-defined node project.

To create any type of user-defined node, use the following steps:

Procedure

1. Create a user-defined node project:
   a) To open the **New User-defined Node Project** wizard, click **File** > **New** > **User-defined Node Project**.
   b) Either choose an existing category, or create a category by clicking **Create a new category**.
      Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project. You can have only one category in each user-defined node project.
   c) Enter the details for the remainder of the fields as required, then click **Finish**.
   A user-defined node project is created.
2. Create a user-defined node:
   a) To open the **New User-defined Node** wizard, click **File** > **New** > **User-defined Node**.
   b) Select the user-defined node project that you created previously.
   c) Select the schema location that you want to host the user-defined node.
      Use a fully qualified schema. Do not use a default or `mqsi` schema.
   d) Enter the details for the remainder of the fields as required.
   e) Select **Implemented as a subflow**. The **Message Flow** editor opens.
   f) Click **Finish**.
3. Decide on the location of your user-defined node in the palette.
   See “Choosing the location of a user-defined node in the palette” on page 2430.

What to do next

You can now define the reusable logic and the terminals and properties of the user-defined node.

Creating a user-defined node from an existing subflow
Create a user-defined node by converting the project for an existing subflow into a user-defined node project, and then creating the user-defined node.

About this task

To create a user-defined node from an existing subflow, use the following steps:

Procedure

1. To convert existing subflows into user-defined nodes, you must convert the project for the subflow into a user-defined node project:
   a) Right-click the existing project, click **Convert to User-defined Node Project**.
      The **Convert to User-defined Node Project** wizard is displayed.
   b) Choose a category from the following options.
      Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project.
      - To use an existing category, select a category from the list provided.
      - To create a new category, select **Create a new category**, enter a name for the category for the nodes that you want to create in the new project.
   c) Enter the details for the remainder of the fields as required.
   d) Click **Finish**.
2. Select the subflow that you want to convert to a user-defined node:
   a) Right-click the subflow to display the menu.
   b) Click **Convert to User-defined Node**.
      The **Convert to User-defined Node** wizard opens.
   c) Enter the details for the remainder of the fields as required.
   d) Click **Finish**.
3. Decide on the location of your user-defined node in the palette.
   See "Choosing the location of a user-defined node in the palette" on page 2430.

Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit

When you are developing a user-defined node in Java or C only, you must create the user interface representation of the node in the IBM App Connect Enterprise Toolkit.

Before you begin

- Perform the steps in "Creating a user-defined node" on page 2383.

About this task

The following topics describe the steps that you must complete:

1. "Creating a user-defined node project" on page 2421
2. "Creating a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2422

What to do next

When you have created the IBM App Connect Enterprise Toolkit representation, test your user-defined node, see "Testing a user-defined node" on page 2430.

Creating a user-defined node project

When you create user-defined nodes, you must first create a user-defined node project to contain the nodes and their supporting files.

About this task

To create a new project for your user-defined node, complete the following steps:

Procedure

1. Click **File** > **New** > **User-defined Node Project**.

   The **New User-defined Node Project** wizard starts.
2. Either select an existing category, or select **Create a new category** and specify the name of a new category for the nodes that you are creating.

   Your user-defined nodes are displayed in the palette in the Message Flow editor under the category you specify for the project.
3. Specify a name for your project.
4. Specify a name for your plug-in identifier. To be consistent with the supplied nodes, and to avoid conflict with the names of node projects that are supplied by other independent software vendors, use your Internet domain name for your organization as part of the name and plug-in identifier. For example, the name and plug-in identifier must be of the form *com.xyz.nodegroup*, where *com.xyz* is the company Internet domain name.

   You can save any number of nodes in a single project.
5. Accept all default values and click **Finish**.

Results

A project folder that contains all the supporting files that are required for your user-defined node is shown in the Application Development view under Independent Resources.

What to do next

Create the user-defined node plug-ins, see "Creating a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2422.

Creating a user-defined node in the IBM App Connect Enterprise Toolkit

Create the representation of a user-defined node created in Java and C only, in the IBM App Connect Enterprise Toolkit.

Before you begin

Complete the following task: "Creating a user-defined node project" on page 2421.

About this task

To create the visual representation of your user-defined node in the IBM App Connect Enterprise Toolkit, complete the following tasks:

1. "Creating the user-defined node plug-in files" on page 2422
2. "Defining the node properties" on page 2423
3. Optional: "Adding help to the node" on page 2427
4. Optional: "Creating node icons" on page 2428
5. Optional: "Adding a property editor or compiler" on page 2428

When you have created the representation of the node, you cannot move it to another folder.

Creating the user-defined node plug-in files

Create the files that contain the message processing logic, for user-defined nodes created in Java and C only.

Before you begin

Complete the task following task: "Creating a user-defined node project" on page 2421.

About this task

Procedure

1. Launch the wizard by clicking **File** > **New** > **User-defined Node**.

   The **New User-defined Node** window opens.
2. Select the parent folder for the node from the list of names that are displayed.

   This folder is the project that you created to contain this node.
3. Specify a schema for this node.

   You must not use the default schema or any other common schema, such as `mqsi`.
4. Specify a name for the node.

   The name must be the name of the node, excluding the `Node` at the end.

   For example, if you have created a node called `BasicNode`, the node name must be `Basic`.
5. Click **Finish**.

   A `.msgnode` file for the new node is created and is added to the project in the Application Development view. The `.msgnode` file is opened in the **Message Node** editor.

What to do next

When you have completed this task, define the node properties, see "Defining the node properties" on page 2423.

Defining the node properties

Define the properties for a user-defined node created in Java or C only, and add input and output terminals so that you can connect it to other nodes in a message flow.

Before you begin

Complete the following tasks:

1. "Creating a user-defined node project" on page 2421
2. "Creating the user-defined node plug-in files" on page 2422

About this task

When you complete the task described in "Creating the user-defined node plug-in files" on page 2422, a `.msgnode` file is created for the new node, and is opened in the **Message Node** editor of the Integration Development perspective. You can now add terminals and properties to the node.

Adding terminals to the node

Procedure

1. If the **Terminals** page is not already displayed, click the **Terminals** tab at the bottom of the **Message Node** editor.
2. Click **Add** to the right of the **In Terminals** or **Out Terminals** fields to add an input or output terminal.

   You must define at least one input terminal, but output terminals are optional.
3. To rename a terminal, click the terminal name so that it is highlighted and shows a flashing cursor after the name, and enter a new name.
4. If your node supports dynamic input or output terminals, select the appropriate check box.

   Dynamic terminals are terminals that you can add to certain nodes after you have added them to a message flow in the **Message Flow** editor. For more information, see "Message flow node terminals" on page 497.

Procedure

1. Click the **Properties** tab at the bottom of the **Message Node** editor.

   On the **Properties** page, you can set the properties for the node, for example, a database name, a host server name, or a password. The properties that you set here must match the properties that you specified in the user-defined node itself by using the `get` and `set` methods.
2. If the node is an input node, click the node name in the hierarchy to highlight it, and select **Input node**. Select **Use integration node default values** if you want the node to initialize with the default values for the integration node.
3. By default, all properties are grouped under the Basic group. You can add new groups in which to place properties. When your custom node is selected in the IBM App Connect Enterprise Toolkit, each group of properties is rendered as a separate tab in the **Properties** view. To create additional groups of properties, click **Add Property Group**  .
4. To add a simple property, click the name of a property group in the hierarchy to highlight it, and click **Add Simple Property** 

   The new property is added to the hierarchy as a child of the property group. Its name is highlighted so that you can change it. A number of fields are displayed in the **Details** section, where you can configure the property.

   a) Select the correct attribute type: one of the built-in types, or a type to match the list of values that the property can have.

   b) Enter any default values, which are shown in the **Properties** view when the node is included in a message flow.

   c) Optional: If you want to use a custom property compiler, in the **Custom Compiler Class** field, click **Import** and select the class you want to use for your custom compiler. The class must implement the IRuntimePropertyCompiler interface. For more information, see "Adding a property editor or compiler" on page 2428.

   d) Optional: If you want to use a custom property editor, in the **Custom Editor Class** field, click **Import** and select the class you want to use for your custom editor. The class must implement the IPropertyEditor interface. For more information, see "Adding a property editor or compiler" on page 2428.

   e) Specify the system property for each attribute that you define:

   Hidden
   The property is not displayed in the **Properties** view or the **Promote Property** dialog box.

   Read only
   The property is displayed, but cannot be changed.

   Mandatory
   A value is required. The field cannot be left blank. Boolean and enum properties are always mandatory.

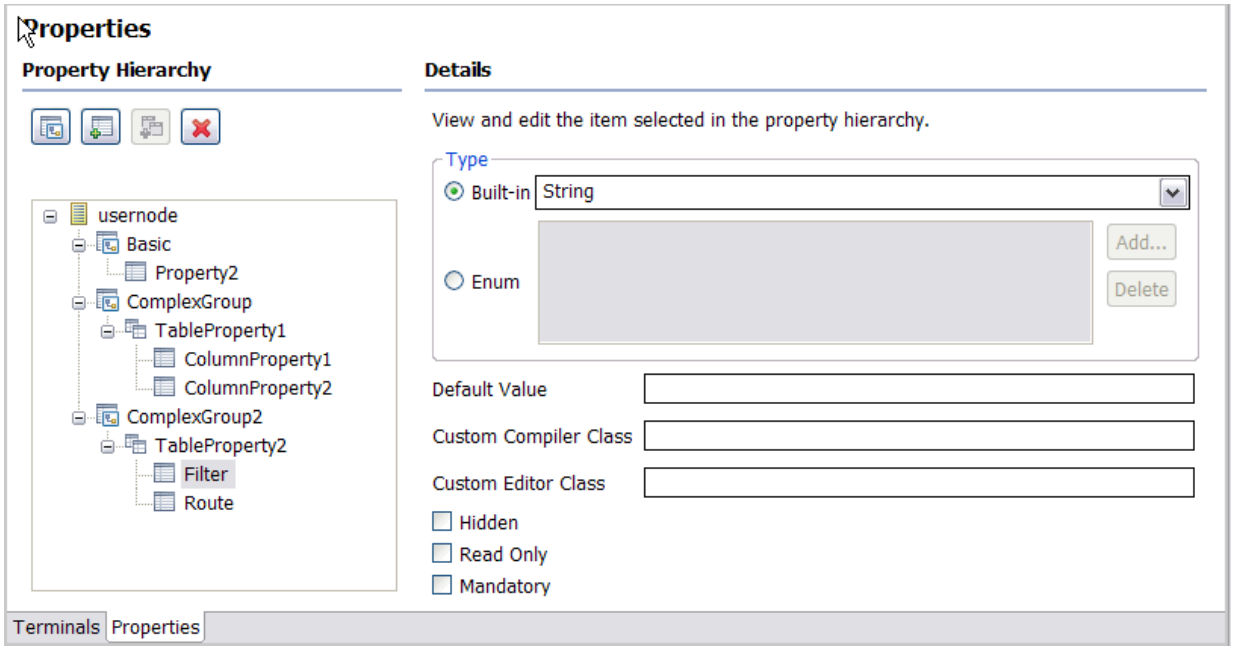
   Configurable
   The property can be configured at deployment time
5. To add a table property, select the name of a property group in the **Properties** view and click **Add Table Property**  .

   In addition to simple properties, a node can also have complex properties. A table property represents a repeating property of a complex type. The new property is added to the hierarchy as a child of the

property group. A number of fields are displayed in the **Details** section where you can configure the table property.

6. To add a column to an existing table, select the name of the table property in the hierarchy, and click **Add Simple Property**  .

   For example, the following figure shows the Property Hierarchy of the usernode, where Filter and Route columns have been added.



   A number of fields are displayed in the **Details** section where you can configure the properties of the column. Define table columns, where each column is assigned to a type.

   a) Select the correct attribute type in the **Type** field: one of the built-in types, or enumeration.

   b) Enter the default value, in the **Default Value** field. This value is shown in the **Properties** view when the node is included in a message flow.

   c) Specify a qualified class name in the **Custom Compiler Class** field for a property compiler. To create a custom compiler, use the IRuntimePropertyCompiler interface.

      For more information about custom property editors and property compilers, see “Adding a property editor or compiler” on page 2428.

   d) Specify a qualified class name in the **Custom Editor Class** field to generate a custom property editor. The property editor specified in this field implements the IColumnPropertyEditor interface responsible for cell editing behavior. Leaving the **Custom Editor Class** field blank means that a

property editor matching-column type is used. Specify your own IColumnPropertyEditor only if you need custom cell editing behavior.

e) Specify the following attributes for each column that you define:

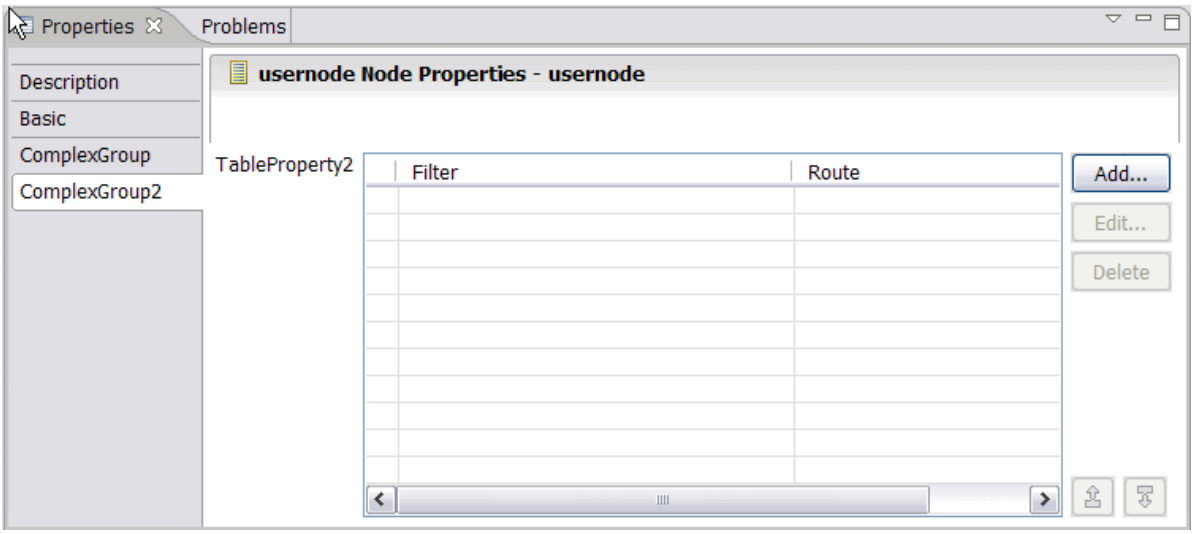
Hidden
Use a hidden column when you want to store, for each row, metadata that is not being displayed.

Read only
The column is displayed, but cannot be changed.

Mandatory
A value is required. The field cannot be left blank. Boolean and enum properties are always mandatory.

Leave the **Custom Editor Class** field of the Details section of a table property blank, unless you want to overwrite the behavior of the entire table. For example, if the table becomes disabled in response to a change in another property editor.

The following figure shows how the Table properties are rendered as a table in the **Properties** view, where you can add, edit, and delete values, and change the order of the values in the table.



7. Optional: Drag the properties in the properties hierarchy to change the order in which they are listed on the properties page.
8. Close the *nodename*`.msgnode` file.
9. Optional: You can customize the text that is displayed in the node properties view for each property.
   To set the text, open the *nodename*`.properties` file and edit the line:

```
Property.propertyName = your descriptive text
```

What to do next

The following tasks are optional:

- “Adding help to the node” on page 2427
- “Creating node icons” on page 2428
- “Adding a property editor or compiler” on page 2428

You can now test your node, see “Testing a user-defined node” on page 2430.

Adding help to the node
Add help information by adding an HTML file to the message flow node that you have created.

Before you begin

Complete the following tasks:

1. “Creating a user-defined node project” on page 2421
2. “Creating the user-defined node plug-in files” on page 2422
3. “Defining the node properties” on page 2423

About this task

Add help information for the message flow node that you have created to explain why and when to use the node, and how it must be configured:

- Topic information that is displayed in the product documentation
- Context-sensitive help that is displayed when you press F1
- Hover help that is displayed when you hold the mouse pointer over the node

All three forms of help are optional; you can create any one or more of the three resources described in the following section.

Procedure

1. Create a `help.html` file in the project to contain the online help that explains what the node does and how you can use it.

   If you have several files, create a separate doc subdirectory in the plug-in project, and store the online help files in that directory.

   You can make the online help for the node look like it is integrated with the product-supplied product documentation, under the leaf node called "User-defined nodes", which you can find in **Reference** > **Message flow development** > **User-defined extensions** > **User-defined nodes**. To make the online help for your node show at that point:

   a) Modify the `plugin.xml` file to include the following extension point to the product documentation:

   ```
   <extension point="org.eclipse.help.toc">
   <toc file="toc.xml"/>
   </extension>
   ```

   b) Create a `toc.xml` file in your user-defined node project, and modify the `link_to` attribute to link to the "UDNodes" anchor that is already defined in the information center contents views:

   ```
   <toc label="My Plugin Node" topic="my_node.htm"
    link_to="../com.ibm.etools.mft.doc/toc.xml#UDNodes">
   <topic label="Mytopic 1" href="topic1.htm>
   </toc>
   ```

   Your help topic is now displayed in the contents views under **Reference** > **Message flow development** > **User-defined extensions** > **User-defined nodes**.

   The sample nodes that are provided with the product demonstrate this option.

   For further explanation of extension points and how to use them, see the PDE guide at Eclipse documentation.

2. Add context sensitive (F1) help to the node. Context-sensitive help is displayed when you click a node in the Integration Development perspective and press F1.

   When a node is created, a `HelpContexts.xml` file is created. This file assigns a context identifier based on the name of the node. Modify the `HelpContexts.xml` file for your node by changing the text in the description field. The name of the `HelpContexts.xml` file must be unique within the

project, but can contain multiple context entries; for example, if you have several nodes within a single project, each node can have its context-sensitive help in the file.

Context-sensitive help is limited in length. A useful way of providing more help to the user is to link from the F1 help to an HTML file that contains further information; for example, to the online help for the node, described previously. Use the following code for the link:

```
<topic href"../plug-in directory/html file" label="Link title">
```

3. Add hover help (known as ToolTip help on Windows) to the node. When you create a user-defined node, a `palette.properties` file is created. Modify this file to contain the hover help for your node, which shows the node name when the palette is not wide enough to display it all.

What to do next

You can add another optional feature, a node icon or a property editor or compiler, or you can test your node, see “Creating node icons” on page 2428, “Adding a property editor or compiler” on page 2428, and “Testing a user-defined node” on page 2430.

Creating node icons
Create the icons that are displayed in the IBM App Connect Enterprise Toolkit when your user-defined node is present.

Before you begin

You must complete the following tasks:

1. “Creating a user-defined node project” on page 2421
2. “Creating the user-defined node plug-in files” on page 2422
3. “Defining the node properties” on page 2423

About this task

You can choose the icon when you create your user-defined node, or you can use the Palette editor to change the icon, see “Choosing the location of a user-defined node in the palette” on page 2430.

What to do next

You can add help, a property editor or compiler, or you can test your node, see “Adding help to the node” on page 2427 or “Adding a property editor or compiler” on page 2428, and “Testing a user-defined node” on page 2430.

Adding a property editor or compiler
Create a property editor by using the IPropertyEditor interface to control how the properties of your user-defined node created in Java or C only, are displayed in the IBM App Connect Enterprise Toolkit. Create a custom compiler by using the IRuntimePropertyCompiler interface; for example, to encrypt a value before sending it to the server.

Before you begin

You must complete the following tasks:

1. “Creating a user-defined node project” on page 2421
2. “Creating the user-defined node plug-in files” on page 2422
3. “Defining the node properties” on page 2423

About this task

The IPropertyEditor interface is used as the basis for all the node property editors in the IBM App Connect Enterprise Toolkit. You can customize the property editor to contain different kinds of controls, such as

text fields and lists. See the IPropertyEditor and IRuntimePropertyCompiler interfaces in the Property editor API.

If you create a custom compiler for your user-defined node, you must install both the compiled runtime files and the user-defined node plug-ins on the integration node to which you want to deploy the node; see "Packaging and distributing user-defined extensions" on page 2449.

If you create a message flow or a subflow that includes a user-defined node and your user-defined node has a custom compiler, be aware of the following information about adding your flow to a BAR file:

- If you want to add your flow to a BAR file as a `.msgflow` file or a `.subflow` file, you must ensure that your custom compiler code implements the IRuntimePropertyCompiler interface. Custom property compilers in versions of WebSphere Message Broker earlier than Version 8.0 use the IPropertyCompiler interface.
- If you want to add your flow to a BAR file as a `.cmf` file, you can continue to use custom compiler code that implements the IPropertyCompiler interface.

For more information about how to add files to a BAR file, see "Adding resources to a BAR file" on page 2470.

Creating a Java class

About this task

To create a Java class for your property editor or compiler, complete the following steps.

Procedure

1. Switch to the Java perspective.
2. Right-click your user-defined node project and click **New** > **Class**. If **Class** is not shown, click **Other**, select **Class** and click **Next**.
3. Type a name for your class in the **Name** field.
4. Complete the following steps, according to whether you are creating a property editor or a property compiler:
   - If you are creating a property editor:

   a) Delete any text in the **Superclass** text field, and click **Browse**.

   b) In the **Choose a type** field, type `AbstractPropertyEditor` and click **OK**.
   AbstractPropertyEditor implements the IPropertyEditor interface.

   - If you are creating a property compiler:

   a) Click **Add** next to the **Interfaces** field.

   b) In the **Choose interfaces** field, type `IRuntimePropertyCompiler` and click **OK**.
5. Click **Finish**.

Testing your property editor or compiler

About this task

To test your property editor, see "Testing a user-defined node" on page 2430.

To test your property compiler, deploy to an integration node the flow that contains your user-defined node.

A custom property editor can use Rational Application Developer or Eclipse APIs. When you migrate to a new version of IBM App Connect Enterprise, your custom property editor might not work if the Rational Application Developer or Eclipse APIs change. Update your property editor code to comply with the changed API.

Choosing the location of a user-defined node in the palette

Use the Palette editor to edit palette-specific information for user-defined nodes, add and delete separators, and rearrange user-defined nodes in the palette.

About this task

- If you change the details for a user-defined node in the Palette editor, save the changes before you add or remove another user-defined node or the changed details will be lost.
- Do not confuse the Palette editor and Customize functions:

  Palette editor
  The Palette editor is started from the Navigator when you select a user-defined node project or the user-defined node virtual folder. The Palette editor is used to set the category, node names, and icons that you want to export so that the user can see the modified node.

  Customize action
  The Customize action is started from the palette in the Message Flow editor when you select a node. The Customize action is used to change the way that the nodes are displayed in the workspace of the current user.

Procedure

1. To open the **Palette** editor use one of the following options:
   - In the Application Development view, under the user-defined node project, select **User Defined Nodes**, click **Edit and Arrange Palette**.
   - In the Application Development view, in the **Other Resources** folder of the user-defined node project, right-click **palette.xmi**. Click **Open With** and select **Palette Editor**.
2. To edit palette-specific information:
   a) In the left side of the **Palette** editor under the Category folder, click the user-defined node that you want to edit. The details about the user-defined node are displayed in the right side of the **Palette** editor.
   b) You can change the details in the **Display Name**, **Tooltip on palette**, and **Node Icons** fields, but you cannot change the **Node Implementation** section.
   c) You can change the category by selecting the Category folder on the left side of the Palette editor, and on the right side either choose an existing category, or rename your own category.
3. To group the user-defined nodes you can add or delete a separator:
   - To add a separator in a user-defined node:
     a. In the left side of the **Palette** editor, right-click the node.
     b. Select **Add Separator Below**. A separator is added below the node in the left side of the **Palette** editor.
   - To delete a separator:
     a. In the left side of the **Palette** editor, right-click **Separator**.
     b. Click **Delete Separator**.
4. To rearrange your user-defined nodes, in the left side of the **Palette** editor either drag the node to its new position, or right-click the node and select **Move Up** or **Move Down**.

Testing a user-defined node

When you have created and installed the required resources, you can test your user-defined node.

Before you begin

Complete the following tasks:

- “Creating a user-defined extension in C” on page 2384 or “Creating a user-defined extension in Java” on page 2403
- “Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit” on page 2421
- “Installing user-defined extension runtime files on an integration node” on page 2455

This topic is for user-defined nodes created in Java and C only.

For user-defined nodes created from a subflow, see “Testing a subflow user-defined node project” on page 2433.

Procedure

Use the following steps to test your user-defined node:

1. To test these projects and resources you must launch a second IBM App Connect Enterprise Toolkit by using the following steps:
   a) Right-click the user-defined node project.
      The menu opens.
   b) Click **Run in New Workbench**.
      - If the project contains errors, you receive an error message that gives you the option to continue, or to cancel the action.
      - If the project is error-free, the **Workspace Launcher** window opens.
        i) You can choose to use a new or an existing workspace. To choose an existing workspace, click **Browse**. The second workspace opens.
        ii) You can now test by using the user-defined nodes in all error-free user-defined node projects in your first workbench.
2. Open the Message Flow editor in the second instance of the IBM App Connect Enterprise Toolkit.
   Your new nodes are displayed in the node palette.
3. Create a message flow that includes your node.
   See “Adding a message flow node” on page 618.
4. Deploy the message flow to an integration node.
   See “Deployment rules and guidelines” on page 2465.
5. Send a test message through the flow and look for the results that you expect (for example, a message put to a target queue).
   You might have to write an application to send the test message to the message flow.

6. Use the diagnostic tools that are provided to determine whether your node works, or if not, what went wrong:
   a) For a description of some common problems and their solutions.
      See “Resolving problems with user-defined extensions” on page 3017.
   b) Write entries to the Administration log from your node.
      See “Using error logging from a user-defined extension” on page 2458.
   c) Switch on user trace at debug level.
      See “Using trace” on page 3026.

      The following debug messages are generated by a user trace to help you to understand the execution of your user-defined nodes and parsers:

      - BIP2233 and BIP2234: A pair of messages traced before and after a user-defined extension implementation function is started. These messages report the input parameters and the returned value.

        In these messages, an "implementation function" can be interpreted as either a C implementation function or a Java implementation method.
      - BIP3904: A message traced before starting the Java `evaluate()` method of a user-defined node.
      - BIP3905: A message traced before starting the C `cniEvaluate()` implementation function (iFpEvaluate member of CNI_VFT) of a user-defined node.
      - BIP4142: A debug message that is traced when starting a user-defined node utility function, where the utility function alters the state of a syntax element. All utility functions that start with `cniSetElement*`, where * represents all nodes with that stem, are included.
      - BIP4144 and BIP4145: A pair of messages traced by certain implementation functions that, when started by a user-defined extension, can modify the internal state of an object in the integration node. Possible integration node objects include syntax element, node, and parser.

        In these messages, an "implementation function" can be interpreted as either a C implementation function or a Java implementation method.
      - BIP4146: A debug message that is traced when starting a user-defined parser utility function, where the utility function alters the state of a syntax element. All utility functions that start with `cpiSetElement*`, where * represents all nodes with that stem, are included.
      - BIP4147: An error message that is traced when a user-defined extension passes an invalid input object to a user-defined extension utility API function.
      - BIP4148: An error message that is traced when a user-defined extension damages an object in an integration node.
      - BIP4149: An error message that is traced when a user-defined extension passes an invalid input data pointer to a user-defined extension utility API function.
      - BIP4150: An error message that is traced when a user-defined extension passes invalid input data to a user-defined extension utility API function.
      - BIP4151: A debug message that is traced when `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to an unexpected value. Expected values are

CCI_SUCCESS, CCI_ATTRIBUTE_UNKNOWN, and CCI_BUFFER_TOO_SMALL. Any other value is an unexpected value.

- BIP4152: A debug message that is traced in the following situations:

  i) `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to CCI_BUFFER_TOO_SMALL.

  ii) `cniGetAttribute2` or `cniGetAttributeName2` is called again with the correct size buffer, however the return code is set to CCI_BUFFER_TOO_SMALL.

d) Add a Trace node to your message flow, and check the output that is generated.

e) Use the flow debugger to debug the flow that contains your node.

See "Testing and troubleshooting message flows" on page 647.

What to do next

When your node behavior is complete and correct, add the new node into your normal palette of nodes in the **Message Flow** editor, see "Packaging and distributing a user-defined node project" on page 2452. Until you complete this task, the new nodes are available only in your test IBM App Connect Enterprise Toolkit session on your local system.

Testing a subflow user-defined node project
Test how the user-defined node is displayed, and how it behaves in your environment.

About this task

You can test how the user-defined node is displayed, and how it behaves in your environment by simulating the node. The user-defined node is displayed in the palette in the **Message Flow** editor, and any change to the plug-in during the simulation is effective immediately; you are not required to restart the simulation.

- To start the simulation of a user-defined node:

  1. Right-click the user-defined node project, click **Start Simulation**. This action adds all the user-defined nodes that are in the project to the palette in the **Message Flow** editor under the category that you defined for the project.
  2. You can drag these nodes from the palette onto the **Message Flow** editor canvas in the same way as the built-in nodes.

- To stop the simulation of a user-defined node:

  1. Right-click the user-defined node project, click **Stop Simulation**.
  2. If any message flows are using the user-defined node, an error is generated because the node is no longer available, and the following message is displayed:

     ```
     User-defined nodes that are contributed by the user-defined node
     project Varname will be removed from the Message Flow editor palette.
     If a copy of these nodes is not available from the plug-in space,
     any message flows that use these nodes might get unresolved node
     errors.
     ```

- If you are using custom editors, compilers, or Java code, these resources cannot be tested in simulation mode. To test and debug these projects and resources you must launch a second IBM App Connect Enterprise Toolkit by using the following steps:
  1. Right-click the user-defined node project. The menu opens.
  2. Click **Run in New Workbench**.
     - If the project contains errors, you receive an error message that gives you the option to continue, or to cancel the action.
     - If the project is error-free, the **Workspace Launcher** window opens.
       a. You can choose to use a new or an existing workspace. To choose an existing workspace, click **Browse**. The second workspace opens.
       b. You can now test and debug by using the user-defined nodes in all error-free user-defined node projects in your first workbench.
- In previous versions of the IBM App Connect Enterprise Toolkit, you were able to use launch configurations to test custom editors, compilers, or Java code. However, launch configurations are not supported with user-defined nodes that are created from a subflow. To test custom editors, compilers, and Java code with user-defined nodes that are created from a subflow, right-click the user-defined node project and click **Run in New Workbench**.

Debugging the message flow in simulation mode
Compile, deploy, test, and debug the message flow that includes your user-defined node.

About this task

You can compile, deploy, test, and debug the message flow that includes your user-defined node in the same way that you test any regular subflow. However, you must set a breakpoint so that the debugger stops at that point, see "Working with breakpoints in the flow debugger" on page 669.

You can debug the message flow only if the following constraints are met:

- The user-defined node project must be in the workspace.
- The user-defined node project must be in the simulation mode.
- The subflow cannot be debugged after it has been exported as a plug-in, unless the source code is in your workspace.

Developing user-defined parsers

Create a user-defined parser to interpret messages with a different format and structure.

Before you begin

Read the following topics:

- "Why use a user-defined extension?" on page 2322
- "User-defined parsers" on page 2345

About this task

A loadable implementation library, or a *LIL*, is the implementation module for a C parser (or node). A LIL is a Linux or UNIX shared object or Windows dynamic link library (DLL), that does not have the file extension `.dll` but `.lil`.

The implementation functions that you must write are listed in C parser implementation functions. The utility functions that are provided by IBM App Connect Enterprise to help you are listed in C parser utility functions.

IBM App Connect Enterprise provides the source for a sample user-defined parser called BipSampPluginParser.c. This example is a simple pseudo-XML parser that you can use in its current state, or you can modify.

The task of writing a parser varies considerably according to the complexity of the bit stream to be parsed. Only the basic steps are described here:

1. “Declaring and defining the parser” on page 2435
2. “Creating an instance of the parser” on page 2436
3. “Deleting an instance of the parser” on page 2437

Declaring and defining the parser

About this task

To declare and define a user-defined parser to the integration node, you must include an initialization function, `bipGetParserFactory`, in your LIL. The following steps outline how the integration node calls your initialization function and how your initialization function declares and defines the user-defined parser:

The following procedure shows you how to declare and define your parser to the integration node:

Procedure

1. The initialization function, `bipGetParserFactory`, is called by the integration node after the LIL has been loaded and initialized by the operating system. The integration node calls this function to understand what your LIL is able to do, and how it must be called. For example:

```
CciFactory LilFactoryExportPrefix * LilFactoryExportSuffix
bipGetParserFactory()
```

2. The `bipGetParserFactory` function calls the utility function `cpiCreateParserFactory`. This function passes back a unique factory name (or group name) for all the parsers that your LIL supports. Every factory name (or group name) passed back must be unique throughout all the LILs in the integration node.
3. The LIL calls the utility function `cpiDefineParserClass` to pass the unique name of each parser, and a virtual function table of the addresses of the implementation functions.

   For example, the following code declares and defines a single parser called InputxParser:

```
{
 CciFactory* factoryObject;
 int rc = 0;
 CciChar factoryName[] = L"MyParserFactory";
 CCI_EXCEPTION_ST exception_st;

 /* Create the Parser Factory for this parser */
 factoryObject = cpiCreateParserFactory(0, factoryName);
 if (factoryObject == CCI_NULL_ADDR) {

  /* Any local error handling can go here */
 }
 else {
  /* Define the parsers supported by this factory */
  static CNI_VFT vftable = {CNI_VFT_DEFAULT};

 /* Setup function table with pointers to parser implementation functions */
 vftable.iFpCreateContext            = cpiCreateContext;
 vftable.iFpParseBufferEncoded       = cpiParseBufferEncoded;
 vftable.iFpParseFirstChild          = cpiParseFirstChild;
 vftable.iFpParseLastChild           = cpiParseLastChild;
 vftable.iFpParsePreviousSibling     = cpiParsePreviousSibling;
 vftable.iFpParseNextSibling         = cpiParseNextSibling;
 vftable.iFpWriteBufferEncoded       = cpiWriteBufferEncoded;
 vftable.iFpDeleteContext            = cpiDeleteContext;
 vftable.iFpSetElementValue          = cpiSetElementValue;
 vftable.iFpElementValue             = cpiElementValue;
 vftable.iFpNextParserClassName      = cpiNextParserClassName;
```

```
 vftable.iFpSetNextParserClassName   = cpiSetNextParserClassName;
 vftable.iFpNextParserEncoding       = cpiNextParserEncoding;
 vftable.iFpNextParserCodedCharSetId = cpiNextParserCodedCharSetId;

 cpiDefineParserClass(0, factoryObject, L"InputxParser", &vftable);
 }

 /* Return address of this factory object to the integration node */
 return(factoryObject);
}
```

The initialization function must create a parser factory by starting `cpiCreateParserFactory`. The parser classes supported by the factory are defined by calling `cpiDefineParserClass`. The address of the factory object (returned by `cpiCreateParserFactory`) must be returned to the integration node as the return value from the initialization function.

For example:

a. Create the parser factory using the `cpiCreateParserFactory` function:

```
   factoryObject = cpiCreateParserFactory(&rc, constParserFactory);
```

b. Define the classes of message supported by the factory using the `cpiDefineParserClass` function:

```
if (factoryObject) {
   cpiDefineParserClass(&rc, factoryObject, constPXML, &vftable);
  }
else {
    /* Error: Unable to create parser factory */
  }
```

c. Return the address of this factory object to the integration node:

```
  return(factoryObject);
}
```

Creating an instance of the parser

About this task

When the integration node has received the table of function pointers, it calls the function `cpiCreateContext` for each instantiation of the user-defined parser. If you have three message flows that use your user-defined parser, your `cpiCreateContext` function is called for each of them. This function allocates memory for that instantiation of the user-defined parser to hold the values for the configured attributes. For example:

Procedure

1. Call the `cpiCreateContext` function:

```
CciContext* _createContext(
  CciFactory* factoryObject,
  CciChar*    parserName,
  CciNode*    parserObject
){
  static char* functionName = (char *)"_createContext()";
  PARSER_CONTEXT_ST* p;
  CciChar          buffer[256];
```

2. Allocate a pointer to the local context and clear the context area:

```
  p = (PARSER_CONTEXT_ST *)malloc(sizeof(PARSER_CONTEXT_ST));

  if (p) {
     memset(p, 0, sizeof(PARSER_CONTEXT_ST));
```

3. Save the parser object pointer in the context:

```
   p->parserObject = parserObject;
```

4. Save the parser name:

```
 CciCharNCpy((CciChar*)&p->parserName, parserName, MAX_NODE_NAME_LEN);
```

5. Return the parser context:

```
return (CciContext*) p;
```

Deleting an instance of the parser

About this task

Parsers are destroyed when a message flow is deleted or redeployed, or when the integration server process is stopped (using the **mqsistop** command). When a parser is destroyed, it must free any used memory and release any held resources using the `cpiDeleteContext` function. For example:

```
void cpiDeleteContext(
  CciParser*  parser,
  CciContext* context
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;

  return;
}
```

Extending the capability of a C user-defined parser

When you have created a C parser, you can extend its capability.

Before you begin

Ensure that you have read and understood the following topic:

- “Developing user-defined parsers” on page 2434

About this task

You can extend the capability of a C parser in the following ways:

- “Implementing the parser functionality” on page 2437
- “Implementing input functions” on page 2438
- “Implementing parse functions” on page 2439
- “Implementing output functions” on page 2439
- “Implementing a message header parser” on page 2440

Implementing the parser functionality

About this task

A parser needs to implement the following types of implementation function:

1. Input functions
2. Parse functions
3. Output functions

Implementing input functions

About this task

Your parser must implement one, and only one, of the following input functions:

- cpiParseBuffer
- cpiParseBufferEncoded
- cpiParseBufferFormatted

The integration node invokes the *input* function when your user-defined parser is required to parse an input message. The parser must tell the integration node how much of the input bitstream buffer that it claims to own. In the case of a fixed-size header, the parser claims the size of the header. If the parser is intended to handle the whole message, it claims the remainder of the buffer.

For example:

1. The integration node invokes the `cpiParseBufferEncoded` input function:

```
int cpiParseBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;
```

2. Get a pointer to the message buffer and set the offset using the `cpiBufferPointer` utility function:

```
  pc->iBuffer = (void *)cpiBufferPointer(&rc, parser);
  pc->iIndex = 0;
```

3. Save the format of the buffer:

```
  pc->iEncoding = encoding;
  pc->iCcsid = ccsid;
```

4. Save the size of the buffer using the `cpiBufferSize` utility function:

```
  pc->iSize = cpiBufferSize(&rc, parser);
```

5. Prime the first byte in the stream using the `cpiBufferByte` utility function:

```
  pc->iCurrentCharacter = cpiBufferByte(&rc, parser, pc->iIndex);
```

6. Set the current element to the root element using the `cpiRootElement` utility function:

```
  pc->iCurrentElement = cpiRootElement(&rc, parser);
```

7. Reset the flag to ensure parsing is reset correctly:

```
  pc->iInTag = 0;
```

8. Claim ownership of the remainder of the buffer:

```
  return(pc->iSize);
}
```

Implementing parse functions

About this task

General parse functions (for example, **cpiParseFirstChild**) are those invoked by the integration node when the syntax element tree needs to be created in order to evaluate an ESQL or Java expression. For example, a Filter node uses an ESQL field reference in an ESQL expression. This field reference must be resolved in order to evaluate the expression. Your parser's general parse function is called, perhaps repeatedly, until the requested element is either created, or is known by the parser not to exist.

For example:

```
void cpiParseFirstChild(
  CciParser*  parser,
  CciContext* context,
  CciElement* element
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc;

  if ((!cpiElementCompleteNext(&rc, element)) &&
      (cpiElementType(&rc, element) == CCI_ELEMENT_TYPE_NAME)) {

    while ((!cpiElementCompleteNext(&rc, element))     &&
           (!cpiFirstChild(&rc, element)) &&
           (pc->iCurrentElement))
    {
      pc->iCurrentElement = parseNextItem(parser, context, pc->iCurrentElement);
    }
  }
  return;
}
```

Implementing output functions

About this task

Your parser must implement one, and only one, of the following output functions:

- cpiWriteBuffer
- cpiWriteBufferEncoded
- cpiWriteBufferFormatted

The integration node invokes the *output* function when your user-defined parser is required to serialize a syntax element tree to an output bit stream. For example, a Compute node might have created a tree in the domain of your user-defined parser. When a node, such as an MQOutput node, needs to serialize this tree, the parser is responsible for appending the output bitstream buffer with data that represents the tree that has been built.

For example:

```
int cpiWriteBufferEncoded(
  CciParser*  parser,
  CciContext* context,
  int         encoding,
  int         ccsid
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                initialSize = 0;
  int                rc = 0;
  const void* a;
  CciByte b;

  initialSize = cpiBufferSize(&rc, parser);
  a = cpiBufferPointer(&rc, parser);
  b = cpiBufferByte(&rc, parser, 0);

  cpiAppendToBuffer(&rc, parser, (char *)"Some test data", 14);

  return cpiBufferSize(0, parser) - initialSize;
```

```
}
```

Implementing a message header parser

About this task

Typically, the incoming message data is of a single message format, therefore one parser is responsible for parsing the entire contents of the message. The class name of the parser that is needed is defined in the `Format` field in the MQMD or the MQRFH2 header of the input message.

However, the message might consist of multiple formats, for example where there is a header in one format followed by data in another format. In this case, the first parser has to identify the class name of the parser that is responsible for the next format in the chain, and so on. In a user-defined parser, the implementation function `cpiNextParserClassName` is invoked by the integration node when it navigates down a chain of parser classes for a message that is composed of multiple message formats.

If your user-defined parser supports parsing a message format that is part of a multiple message format, the user-defined parser *must* implement the `cpiNextParserClassName` function.

For example:

1. Call the **`cpiNextParserClassName`** function:

```
void cpiNextParserClassName(
  CciParser*  parser,
  CciContext* context,
  CciChar*    buffer,
  int         size
){
  PARSER_CONTEXT_ST* pc = (PARSER_CONTEXT_ST *)context ;
  int                rc = 0;
```

2. Copy the name of the next parser class to the integration node:

```
  CciCharNCpy(buffer, pc->iNextParserClassName, size);

  return;
}
```

Compiling a C user-defined extension

Compile user-defined extensions in C for all supported operating systems.

Before you begin

If you create your own user-defined nodes, parsers, and user exits in C, compile them on the operating system on which the target integration node is running. Samples are provided for both nodes and parsers, and are described in Sample node files and Sample parser files. Use the instructions here to compile the samples. If you want to create your own extensions, see the following topics:

- “Creating a user-defined extension in C” on page 2384
- “Developing user-defined parsers” on page 2434
- “Developing user-defined exits” on page 2443

About this task

These instructions use the file names of the supplied samples. If you are compiling your own user-defined extensions, substitute your own file names.

When you compile a user-defined extension that is written in C, you need a compatible compiler. For details of supported compilers, see IBM App Connect Enterprise system requirements.

Header files

About this task

The following header files define the C interfaces:

BipCni.h
Message processing nodes

BipCpi.h
Message parsers

BipCci.h
Interfaces common to both nodes and parsers

BipCos.h
Platform-specific definitions

Compiling

About this task

Compile the source for your user-defined extension on each of the supported operating systems to create the executable file that the integration node calls to implement your user-defined extension. On Linux and AIX and systems, this file is a loadable implementation library (LIL) file; on Windows systems, it is a dynamic load library (DLL) file.

The libraries that you build to contain user-defined nodes or parsers must have the extension `.lil` on all operating systems so that the integration node can load them. Libraries that contain user exits must have the extension `.lel` on all operating systems. The examples in this topic show libraries with the extension `.lil`

All extensions must be compiled as 64-bit libraries.

Refer to the documentation for the compiler that you are using for full details of available compile and link options that might be required for your programs.

Navigate to the directory where your user-defined extension source code is located, and follow the instructions for your operating system:

- AIX
- Linux
- Windows

Compiling on AIX

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

The following instructions are for compiling an extension:

```
xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c SwitchNode.c \
   -o SwitchNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c TransformNode.c \
   -o TransformNode.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
```

```
   -c BipSampPluginUtil.c \
   -o BipSampPluginUtil.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c Common.c \
   -o Common.o

xlc_r -q64 \
   -I. \
   -I/install_dir/server/include/plugin \
   -c NodeFactory.c \
   -o NodeFactory.o

xlc_r -q64 \
      -qmkshrobj \
      -bM:SRE \
      -bexpall \
      -bnoentry \
      -o SwitchNode.lil SwitchNode.o \
         BipSampPluginUtil.o Common.o NodeFactory.o \
      -L /install_dir/server/lib \
      -l imbdfplg

chmod a+r SwitchNode.lil
```

Compiling on Linux

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

When you compile programs on Linux on POWER, replace the option `-fpic` with `-fPIC` if you want to use dynamic linking and avoid any limit on the size of the global offset table.

The following instructions are for compiling an extension on Linux on x86-64, Linux on POWER, and Linux on Z. To compile the extension for Linux on POWER, include the **-02** parameter in the compile and link examples, and include the **-fexceptions** parameter in the compile examples.

```
g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   TransformNode.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   SwitchNode.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   BipSampPluginUtil.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX  -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   Common.c

g++ -c -m64 -Wall -Wno-format-y2k -fpic \
   -I. \
   -I/install_dir/server/include \
   -I/install_dir/server/include/plugin \
   -DLINUX -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT \
   NodeFactory.c

g++ -m64 -o samples.lil \
```

```
   TransformNode.o \
   SwitchNode.o \
   BipSampPluginUtil.o \
   Common.o NodeFactory.o \
   -shared -lc -lnsl -ldl \
   -L/install_dir/server/lib -limbdfplg
```

These commands create the file `samples.lil` that provides TransformNode and SwitchNode objects.

Compiling on Windows

About this task

When you compile a user-defined extension that is written in C, use a supported compiler.

Ensure that you include a space between `SwitchNode.c` and `BipSampPluginUtil.c`, and also between `-link` and `/DLL`.

Enter the command as a single line of input; in the following example the lines are split to improve readability.

```
cl /VERBOSE /LD /MD /Zi /EHsc /I.
   /Iinstall_dir\server\include\plugin
   SwitchNode.c BipSampPluginUtil.c Common.c
   NodeFactory.c TransformNode.c
   -link /DLL install_dir\server\lib\imbdfplg.lib
   /OUT:SwitchNode.lil
```

If you have correctly set the *LIB* environment variable, you do not have to specify the full paths to the LIB files.

Developing user-defined exits

You can develop and deploy a user-defined exit.

Before you begin

- Read "User exits" on page 2349.

About this task

The following topics describe the steps required to develop and deploy a user-defined exit.

Procedure

1. "Developing a user exit" on page 2443
2. "Deploying a user exit" on page 2444
3. "Transaction tracking user exit sample" on page 2446

Developing a user exit

You can develop a user exit by declaring it, implementing its behavior, and then compiling it.

About this task

A user exit is user-provided custom software, which is written in C, which can be used to track data that passes through message flows. For an example of how you might implement a user exit, see "Transaction tracking user exit sample" on page 2446.

Procedure

To develop a user exit, complete the following steps:

1. Declare the user exit.

   Declare a user exit by using the `bipInitializeUserExits` function to specify the following properties:

   a) Name (used to register and control the active state of the exit)

   b) User context storage

   c) A function to be invoked (for one or more Event Types)

2. Implement the user exit behavior.

   When the user exit is declared, a set of functions is registered, and these functions are invoked when specific events occur. The behavior of the user exit is provided by implementing these functions. The following table lists the events and their associated functions:

Event	Function
A message is dequeued from the input source.	`cciInputMessageCallback`
A message is propagated to the node for processing.	`cciPropagatedMessageCallback`
A request message is sent to the output node's transport, and transport-specific destination information is written to "WrittenDestination" in the LocalEnvironment.	`cciOutputMessageCallback`
The node completes processing.	`cciNodeCompletionCallback`
The transaction ends	`cciTransactionEventCallback`

3. Your user exit code must implement the cleanup function.

   The user exit library must implement the `bipTerminateUserExits` function. This function is invoked as the integration server's process is ending, and your user exit must clear up any resources that are allocated during the `bipInitializeUserExits` function.

4. Compile.

   Use your existing process for your environment to compile your user exit. See “Compiling a C user-defined extension” on page 2400 for more details.

   For more information about supported C compilers, see IBM App Connect Enterprise system requirements.

5. Link the compiled code to a library with the extension `.lel` that exports the `bipInitializeUserExits` and `bipTerminateUserExits` functions.

What to do next

Deploy the user exit by following the instructions in “Deploying a user exit” on page 2444.

Deploying a user exit

Deploy your user exit to the integration node.

Before you begin

- Write and compile the user exit code. See “Developing a user exit” on page 2443.
- Ensure that the exit:

  1. Is in a library that has the extension `.lel`
  2. Exports the functions `bipInitializeUserExits` and `bipTerminateUserExits`

About this task

You can set the state of the user exit dynamically to active, or inactive, on a per-message flow basis without restarting the integration node.

You can set the state of the user exit dynamically to active, or inactive, for an independent integration server or an integration server that is managed by an integration node. You do not have to restart the integration node.

You can set the state of the user exit dynamically to active, or inactive, for an integration node. You must start the integration node.

To deploy the user exit:

Procedure

1. Install the user exit code on an integration node.

   The library that contains the user exit code must be installed on a file system that can be accessed by the integration node. For example, the file must have read and execute authority for the user ID under which the integration node runs. The integration node looks in the following places for libraries that contain user exits:

   - The `userExitPath` property of the integration node.

     The integration node property `userExitPath` defines a list of directories that are separated by colons (semicolons on Windows). You can set this property by using one of the following methods:

     – Set the `userExitPath` property when you create a new integration node, use the `-x` flag on the **mqsicreatebroker** command.

     – Set the `userExitPath` property for an existing integration node, use the `-x` flag on the **mqsichangeflowuserexits** command.

     – Set the `userExitPath` property for an existing integration node by modifying the entry in the `node.conf.yaml` file for the integration node.

   - The environment variable *MQSI_USER_EXIT_PATH*.

     Append the directory that contains extension files to the environment variable *MQSI_USER_EXIT_PATH* associated with the environment in which the integration node is running.

   If the `userExitPath` property and the environment variable *MQSI_USER_EXIT_PATH* are both set, the environment variable takes precedence. All the directories in the environment variable are searched in the order in which they appear in the variable. Then, all the directories in the integration node property are searched in the order in which they appear in the property.

2. Load the user exit library into the integration node's processes.

   When the user exit library is installed on the integration node, you must load it in one of the following ways:

   - Stop and restart the integration node.
   - Run the **mqsireload** command to restart the integration server processes.

3. Activate the user exit.

   User exits can be active or inactive, and are inactive by default. You can change the state of a user exit dynamically by updating the node.conf.yaml file, or by using the **mqsichangeflowuserexits** command. You must restart the integration node.

   To set the default user exit state for an integration node:

   a) Stop the integration node.

   b) Set the `--active-user-exit-list` property of the integration node by using one of the following methods:

      - Modify the `activeUserExitList` property in the `node.conf.yaml` file.
      - Use the `-x` flag on the **mqsichangeflowuserexits** command.

   c) Start the integration node and check the system log to ensure that all integration servers start without error.

      If you specify any user exit names that are not provided by any library that is loaded by the integration server, a BIP2314 message is written to the system log. All flows that are in the integration servers fail to start.

   d) Optional: If a BIP2314 message is written to the system log and the flows in the integration servers fail to start, take one of the following actions:

      - Provide a library in the user exit path that implements the exit. Then, run the **mqsireload** command, or restart the integration node, to load an exit from the library.
      - Run the **mqsichangeflowuserexits** command to remove the exit from both the active and inactive lists.

   You can also override the default user exit state for an integration node. You can use the **mqsichangeflowuserexits** command to activate, or deactivate, user exits on a per-integration node, per-integration server, or per-message flow basis. The order of precedence is message flow then integration server, then integration node. When multiple exits are active for a flow, the integration node starts them in the order that is defined by the **mqsichangeflowuserexits** command.

Transaction tracking user exit sample

The AceSpanTraceExample user exit sample demonstrates how you can use a user exit to capture flow context information, which could then be used to implement a simple span trace for message flows that contain HTTP, Kafka, or MQ transport nodes.

A span trace can be used to track transactions across multiple components, so that you can see how the transactions have been routed through the components. This approach typically passes a span identifier from one component to another, through transport headers, to allow the transaction to be correlated; this enables you to see a broader view of the transactions than you could see if you used a separate trace for each individual component.

If you run an integration server with this sample user exit enabled, an MQ queue on the integration server's default queue manager (`AceSpanTraceExample`) is used to write a span summary message each time a flow with an MQ, Kafka, or HTTP transport input node processes a message. The user exit reads or writes a span identifier when the flow inputs or outputs an MQ, Kafka, or HTTP message.

The AceSpanTraceExample sample is in the *`install_dir`*`\sample\extensions\userexits\transports` directory, and the loadable exit library is called `AceSpanTraceExample.lel`.

The AceSpanTraceExample is a C++ implementation that uses the App Connect Enterprise interface defined through the MqsiTransportUserExitBase class, which uses C user exit callbacks to show how a message flow span trace might be implemented. The logic of the exit is implemented in the `AceSpanTraceExample.hpp/cpp`.

The sample implements the following transport-specific callback functions, which are invoked by IBM App Connect Enterprise when HTTP, Kafka, and MQ transport nodes process messages:

- `onTransportInputMessage`

  This callback informs the user exit that an MQInput, HTTPInput, SOAPInput, HTTPAsyncResponse, SOAPAsyncResponse, or KafkaConsumer node has received a message, and signals the start of a message flow processing span. The callback provides the message flow and node context and any requested headers, such as MQ MQRFH2 header fields, HTTP headers, or Kafka properties.

- `beforeTransportOutputMessage`

  This callback informs the user exit that an MQOutput, HTTPRequest, HTTPAsyncRequest, SOAPRequest, SOAPAsyncRequest, RESTRequest, RESTAsyncRequest, or KafkaProducer node is about to emit a message. This callback allows the user exit to set any required headers in the output message, such as MQ MQRFH2 header fields, HTTP headers, or Kafka properties.

- `afterTransportOutputMessage`

  This callback informs the user exit that an MQOutput, HTTPRequest, HTTPAsyncRequest, SOAPRequest, SOAPAsyncRequest, RESTRequest, RESTAsyncRequest, or KafkaProducer node has emitted a message. This callback allows the user exit to receive details of the message emitted and its destination, such as MQ `MQMD.msgid`.

- `onTransportResponseMessage`

  This callback informs the user exit that an HTTPRequest, SOAPRequest, or RESTRequest node has received a response message. The callback provides the message flow and node context and any requested headers.

- `beforeTransportGetMessage`

  This callback informs the user exit that an MQGet or KafkaRead node is about to start a read operation to attempt to get a message.

- `onTransportGetMessage`

  This callback informs the user exit that an MQGet or KafkaRead node has received a message. The callback allows the user exit to obtain required headers from the received message, such as MQ MQRFH2 header fields or Kafka properties.

- `onTransportTransactionComplete`

  This callback informs the user exit that a message flow processing span, which started at an `onTransportInputMessage`, is complete. The callback allows the user exit to access the completion status and any last exception code.

For more information, see the functional description in the `AceSpanTraceExample.hpp` header file.

Note: The following class files, which are in the `Base` subfolder, must **not** be modified:

- MqsiTransportUserExitBase
- MqsiUserExitWrapper

If the transaction tracking exit is enabled for a message flow that includes any of the following message flow nodes, and the output terminal of these nodes has multiple direct connections to downstream message flow nodes (rather than using a FlowOrder node), the `onTransportResponseMessage` or `onTransportGetMessage` callback is invoked each time a message is propagated to the directly connected output connections:

- HTTPRequest
- RESTRequest
- SOAPRequest
- MQGet
- KafkaRead

If you need to include multiple downstream connection branches in the message flow, use a FlowOrder node to control the order in which the connection branches are invoked.

The sample source code is also precompiled into the loadable exit library, `AceSpanTraceExample.lel`, which can be used by the integration server if it is specified in the `UserExits` section of the integration server's `server.conf.yaml` file:

```
UserExits:
  activeUserExitList: 'AceSpanTraceExample'
  userExitPath: '<path>'
```

The user exit path defines the directory or set of directories that the integration server's exit manager inspects to see whether there are any loadable exit library files.

When an integration server starts and finds the sample loadable exit library, it calls its `onStartServer` function and passes context information including the name of the integration server and the version at which the integration server is running, and also allows the sample to register the name of a span trace example. If that name matches an entry in the `activeUserExitList`, the transport-specific callback functions in this library are invoked by the integration server when messages are processed by the message flows.

The `onStartServer` callback in this example also provides a data structure of transports for which it should be called (either HTTP, Kafka, or MQ) and field names and locations that the functions in the user exit library will be passed (on an input message) or set (on an output message).

The exit callbacks in the sample are called from message flow nodes that interact with the MQ, Kafka, or HTTP transport. All these transports receive message flow context information including server name, application name, library name, message flow name, and message flow node name and type. Additionally, they are passed data from fields as defined in the example data structure table. For example, the following table defines the data structure for HTTP:

Table 77. Example data structure for HTTP

Transport	**Location**	**Field name**
HTTP	HTTPHeader	traceparent
HTTP	HTTPHeader	tracestate
HTTP	HTTPHeader	x-remote-addr

With the data structures defined in this table, the transport-specific callback functions implemented in the sample carry out the following processing:

- The `onTransportInputMessage` receives any headers that are found in the input, such as `traceparent`, `tracestate`, and `x-remote-addr`. It sets a new value for `traceparent` and then saves the values in the environment, which can then be accessed from other message flow nodes that are involved in the trace.

  For example, you could have a message flow containing an HTTPInput node, and the `onStartServer` function has registered the data structure and specified that it should be called for HTTP transport messages. When the HTTPInput node receives a message, it checks to see whether it needs to invoke the user exit, based on the information in the transport section. If it does need to invoke the user exit, it performs a lookup in the message assembly to extract the values that are available, then calls the `onTransportInputMessage` function that is implemented in the user exit library. In addition to the data structure, it also passes some flow context information, such as the name of the integration server, the application, the message flow name, the message flow node name and type, and the transport type of the message flow node that has called this user exit.

  The sample implements this user exit interface. The sample receives the headers and sets a new value for `traceparent` (`traceParent = TraceStartedinACE`), which appears in the span context. The sample also sets a value for the `AceTrace` environment variable by using `setEnvironmentValue(`*`name`*`)`, which provides an audit trail of the nodes that have called the user exit.

- The `beforeTransportOutputMessage` function retrieves the value of `traceparent` from the environment by using `getEnvironmentValue(name)`, and appends the name of the current message flow node to the `AceTrace` environment variable. The outbound HTTPOutputHeader is then set (`traceparent=traceparent+"flow-context"`) and the returned values are updated into the output message header.

  For example, you could have an HTTPRequest node that is about to send an output message. The transport type has been checked, and it has been determined that the callback function in the user exit library needs to be invoked. The data that is relevant to the HTTP transport is passed over, and the message flow exit is invoked with the flow context information. The sample then appends the information about the current message flow (such as the message flow name, application name, and message flow node name) into the `AceTrace` environment variable, which is recording the audit trail of nodes that have caused the user exit to be invoked. The sample also retrieves the `traceparent` that was set by the HTTPInput node, and concatenates that with the flow context information. The HTTPRequest node has now been added onto the end of the `traceparent` in the environment, which was previously set by the HTTPInput node. The information in the data structure is updated into the message assembly and set into the outbound HTTP header.
- The `onTransportResponseMessage` allows the user exit to record the arrival of a response message from an HTTP request.
- The `beforeTransportGetMessage` signals that a mid-flow Get or Read message flow node is about to start a get or read operation on the transport.
- The `onTransportGetMessage` signals that a mid-flow Get or Read message flow node has received a message from a transport and provides trace data from it.
- The `onTransportTransactionComplete` message is used by the user exit to complete the span trace and issue a summary message to an MQ queue on the integration server's default queue manager (`AceSpanTraceExample`).
- The `afterTransportOutputMessage` retrieves information from the transport relating to the message that has been transmitted. For HTTP, this would include the HTTP `RequestURL`.

Packaging and distributing user-defined extensions

When you have created and tested a user-defined extension, you can package and distribute it.

Before you begin

Complete the following tasks:

- “Developing user-defined extensions” on page 2321
- “Testing a user-defined node” on page 2430

About this task

When you have created and tested your user-defined extension, you can distribute these resources to other computers.

For user-defined extensions created in Java or C:

1. Package and install the user-defined extensions. To package and install the resources that make up the workbench representation of your user-defined node, see “Packaging and distributing a user-defined node project” on page 2452.
2. Copy the files generated by the compilation step to all the computers on which you have created integration nodes that might need these resources. If you have created a user-defined node in Java or C that uses a custom property compiler, you must also install the user-defined node project on integration nodes to which you want to deploy the node. See “Installing user-defined extension runtime files on an integration node” on page 2455. For a more automated approach, see Installing a user-defined extension to current and past versions of.

For user-defined nodes created from subflows:

- Package and install the user-defined nodes implemented as a subflow, see "Packaging and distributing a user-defined node project" on page 2452.

Packaging a Java user-defined node

How to package a Java user-defined node.

Before you begin

You must have a user-defined node written in Java. This node can be one of the provided sample nodes that are described in Sample node files, or a node that you have created yourself by using the instructions in either "Creating a message processing or output node in Java" on page 2409 or "Creating an input node in Java" on page 2403.

About this task

You can package a user-defined node in two ways:

- **PAR**

  A Plug-in Archive (PAR) is the deployment unit for Java user-defined nodes. The PAR contains the user-defined node classes and, if required as dependencies, can contain JAR files. A PAR file is a compressed file with a `.par` file extension. The directory structure in the `.par` file has the following format:

  - `/classes`

    The user-defined node classes are stored in this location.

  - `/lib`

    JAR files that are required by the user-defined node are stored in this location. This directory is optional because it might not be necessary to include JAR files.

  The following procedure describes how to package an example user-defined node, *parexamplenode*. In this example, the PAR is to be contained in *par.example.parexamplenode.class* with a JAR file dependency *dependency.jar*.

  1. Create the directory structure; for example:
     - `/classes/par/example/parexamplenode.class`
     - `/lib/dep.jar`
  2. Issue a file compression command to create the PAR; for example:

     ```
     jar cvf parexample.par classes lib
     ```

  The PAR must be placed in the LIL path that is specified in "Installing user-defined extension runtime files on an integration node" on page 2455.

- **JAR**

  User-defined nodes can be packaged by using a simple JAR. For example, if your node is defined in `example/jarexamplenode.class`, create the JAR by using the `jar cvf jarexample.jar example` command.

  The preferred way to package a Java user-defined node is to use a PAR file, because all dependencies can be packaged with the node, and each node is loaded in a separate class loader. For information about loading classes, see "User-defined node class loading" on page 2451.

  The JAR must be placed in the LIL path that is specified in "Installing user-defined extension runtime files on an integration node" on page 2455.

Deployment dependencies

About this task

If a user-defined node requires an external package, the package can be deployed in one of following ways:

- The external packages can be added to the `/lib` directory in the deployed PAR.
- For external packages that are shared between several node types, the packages can be added to one of the following locations:
  - One of the `shared-classes` directories. For more details of these directories, see "Java shared classloader" on page 1752.
  - The CLASSPATH environment variable, where all user-defined nodes that are in the integration node installation can access the packages

User-defined node class loading

Details the Java classes packaging options and loading order precedence for user-defined nodes.

Java user-defined node classes can be packaged and loaded in two ways:

- Plug-in Archive (PAR) file; an integration node defined format - Each PAR file is given it's own class loader which isolates any classes it loads from any other part of the IBM App Connect Enterprise, including nodes in other PAR files.
- Standard Java archive (JAR) file - All JAR files containing plug-in nodes are loaded into the same classloader and can access classes in JAR files containing other user-defined nodes.

For both packaging mechanisms, if the classloader cannot find a required class within the package it defers to the shared class loader to find the required class. The shared classloader looks in a set of directories on the integration node machine and loads any JAR files found. It can be used to install any required JAR files that do not need to be repeatedly deployed, such as client libraries that the Java compute nodes need to use. For more details, see "Java shared classloader" on page 1752.

If the required class cannot be found in any of the deployed JAR files, or in the JAR files installed in the shared classes directories, a classloader containing all of the integration node supplied classes is checked (for example: this classloader contains the `jplugin2.jar`), followed by the classpath, and then finally the Java virtual machine (JVM) system classloader.

Two key points must be considered when deciding which of the above mechanisms are used to load a class:

- Isolation between different applications (for example: adding classes to the classpath makes them available to every part of IBM App Connect Enterprise and can cause conflicts).
- Delegation from one classloader to another can only occur in one direction. If a class is resolved in the shared classloader, then it cannot directly create classes in the PAR classloader.

User-defined nodes class loading search paths

User-defined nodes package in a PAR file

The integration node uses the following search path to find user-defined node classes:

1. `/classes` to locate classes in the deployed PAR file.
2. `/lib` to locate any JAR files in the deployed PAR file.
3. *workpath*`/config/`*<my_int_node_name>*`/`*<my_int_server_label>*`/shared-classes` to locate any JAR files in the integration server `shared-classes` directory.
4. *workpath*`/config/`*<my_int_node_name>*`/shared-classes` to locate any JAR files in the integration node `shared-classes` directory.
5. *workpath*`/shared-classes/` to locate any JAR files in the top level `shared-classes` directory.
6. CLASSPATH environment variable.

User-defined nodes package in a JAR file

The integration node uses the following search path to find user-defined node classes:

1. The deployed JAR file.
2. *workpath*/config/*<my_int_node_name>*/*<my_int_server_label>*/shared-classes to locate any JAR files in the integration server shared-classes directory.
3. *workpath*/config/*<my_int_node_name>*/shared-classes to locate any JAR files in the integration node shared-classes directory.
4. *workpath*/shared-classes/ to locate any JAR files in the top level shared-classes directory.

Packaging and distributing a user-defined node project

Export the user-defined node project to make it available for other users.

About this task

To export the user-defined node project, you can package the user-defined node project as plug-in JAR files or as an update site:

- "Packaging as plug-in JAR files" on page 2452
- "Packaging as an update site" on page 2453

What to do next

You can now install the plug-in on all the computers on which your IBM App Connect Enterprise Toolkit users might want to use them, following the instructions in "Installing a user-defined node" on page 620.

If you have created a user-defined node in Java or C that uses a custom property compiler, you must also install the user-defined node plug-in on the integration nodes or independent integration servers onto which you want to deploy the user-defined node; see the following topics:

- "Installing user-defined extension runtime files on an integration server" on page 2456
- "Installing user-defined extension runtime files on an integration node" on page 2455

Packaging as plug-in JAR files

Export the user-defined node project as plug-in JAR files to make it available for other users.

About this task

If you use this option, the user-defined node user must copy the user-defined node plug-ins into the plug-ins folder in the IBM App Connect Enterprise Toolkit. However, the user-defined node user can write a script to automate the copying process, so that the user-defined nodes are deployed automatically.

To package your user-defined node projects so that they are available in the environment of the user, complete the following tasks:

Procedure

1. Right-click the project that you want to package, select **Package**.
   The **Package and Distribute User-Defined Nodes** wizard opens.
2. Select **Plug-in jars**. Click **Next**.
3. Select the plug-ins and fragments that you want to use.
   Projects upon which the subflow user-defined node depends, such as message set projects, are automatically included and are not shown in the list.
4. Navigate through the wizard, completing the fields as required. Click **Finish**.

Packaging as an update site

Create an installation site for Web page distribution of the user-defined node projects that includes or references other projects or plug-ins. You can either create a new update site, or use an existing one.

About this task

To package your user-defined node projects so that they are available in the environment of the user, complete the following tasks:

Procedure

1. Right-click the project that you want to package, select **Package**.
   The **Package and Distribute User-Defined Nodes** wizard opens.
2. Select **Installation package**.
   You can either create a new installation site or use an existing installation site in the workspace.
   - To create a new installation site:
     a. Select **Create a new package**, and in the **Create a new package** field, enter a name for the package. If a project with that name exists in the workspace, you are asked if you want to delete the project so that you can continue with packaging.
     b. Click **Next**.
     c. Optional: Select any projects and plug-ins that you want to include in your package:
        i) The top section shows the user-defined node projects that are available in your workspace. Select the projects that you want to include in your package.
        ii) The bottom section shows all the Eclipse plug-ins that the selected user-defined node projects require. IBM App Connect Enterprise Toolkit built-in plug-ins that have a plug-in identifier that starts with `com.ibm.etools.mft` are not shown. The Eclipse plug-ins that are required are a combination of the following plug-ins:
           – All Eclipse plug-ins that are listed as the dependency plug-ins in the plug-in manifest for the user-defined node project
           – All Eclipse plug-ins that contributed user-defined nodes that are being used by one of your selected user-defined node projects
        iii) Click **Next**.
     d. Optional: Further customize your user-defined node package by adding in any detailed information that you want to include. Click **Next**.
     e. Click **Finish**. When you click **Finish**, the selected user-defined node project and its dependent user-defined node projects in the workspace are packaged into an Eclipse feature, and an Eclipse Update Site is built. The Eclipse feature and Eclipse update site project use the name that you have given to the package appended with `.feature` or `.site`. Any space in the package name is replaced by a dot (.).
     f. The **Distribution** window opens. To save the instructions to the clipboard, click **Save the instruction above to clip board**.
     g. To select a destination for your user-defined node package, in the "**Web distribution**" section, click **Copy project *Your project name*.** . The "**Copy package**" window opens. Select the location, click **OK**.
   - To use an existing installation site you must have already created an installation site.
     a. Select **Update user-defined nodes in an existing package**, select an installation site from the list. Click **Finish**. When you click **Finish**, the selected user-defined node project and its dependent user-defined node projects in the workspace are packaged into an Eclipse feature, and an Eclipse Update Site is built. The Eclipse feature and Eclipse update site project use the

name that you have given to the package appended with `.feature` or `.site`. Any space in the package name is replaced by a dot (.).

b. The **Distribution** window opens. To save the instructions to the clipboard, click **Save the instruction above to clip board**.

c. To select a destination for your user-defined node package, in the "**Web distribution**" section, click **Copy project *Your project name*.** . The "**Copy package**" window opens. Select the location, click **OK**.

Installing from an update site

If you have packaged your user-defined nodes into an update site, use the software update mechanism to install the user-defined nodes.

About this task

To install your user-defined nodes from an update site, use the following steps:

Procedure

1. Click **Help** > **Software Updates**.
   The **Software Updates and Add-ons** window opens.
2. Click the **Available Software** tab.
   A list of available sites is displayed. If the update site that you want to use is not listed, click **Add Site**. The **Add Site** window opens. Select the location that you want to use, click **OK**.
3. In the **Available Software** tab, select the site that you want to use, click **Install**.

Updating and uninstalling from an update site

You can update, uninstall, or revert the configuration of any user-defined node plug-ins that have been previously installed from an update site.

About this task

To update, uninstall, or revert the configuration of any user-defined node plug-ins, complete the following steps:

Procedure

1. Click **Help** > **Software Updates**.
   The **Software Updates and Add-ons** window opens.
2. Click the **Installed Software** tab.
3. Select the previously installed user-defined node plug-in.
4. Click **Update**, **Uninstall**, or **Revert Configuration** depending on the action you want to perform.

Installing user-defined extension runtime files on an integration node

Install the compiled runtime files for your user-defined extension on the integration node on which you want to test its function. If your user-defined node uses a custom compiler, install the user-defined node plug-in to the integration node on which you want to deploy the node.

Before you begin

- Create and compile your user-defined extension using the procedure described in "Compiling a Java user-defined node" on page 2418 or "Compiling a C user-defined extension" on page 2400.
  - The files that have been created for extension created in C depend on the underlying integration node operating system:

    Windows A dynamic link library (DLL), named with a file type of `.lil`.

    Linux A shared object, again with a file type of `.lil`.
  - For Java nodes, a Java Archive file (JAR), with a file type of `.jar` (on all operating systems).
- If you have created a user-defined node, you must also complete the task that is described in "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421.
- If your user-defined node contains a custom compiler, you must package your user-defined node project; see "Packaging and distributing a user-defined node project" on page 2452.

About this task

This task instructs you to stop and restart integration nodes. This action is required in all but the two circumstances described in step "4" on page 2455 later in this section, although if you do stop and restart the integration node, you can ensure that anyone with an interest in a particular integration server is made aware that recent changes have been made.

This task is applicable to user-defined nodes written in Java or C only. If your user-defined node contains a custom compiler, you must install the compiled runtime files and the user-defined node plug-in to the integration node. If your user-defined node does not contain a custom compiler, install only the compiled runtime files to the integration node.

To install runtime and user-defined node plug-in files on the integration node:

Procedure

1. Stop the integration node on which you want to install your compiled or packaged user-defined extension file (files with extension `.lil`, `.jar`, `.par`, `.pdb`, or `.lel`)
2. Create a directory if you do not already have one for this purpose. Add the directory to the LILPATH: set LILPATH=<directory>

   CAUTION: Do not put the `.lil`, `.jar`, `.par`, `.pdb`, or `.lel` files in the IBM App Connect Enterprise installation directory, because they might be overwritten by the integration node.
3. Put your user-defined file in the directory, and make sure that the integration node has access to it.

   For example, on Linux use the **chmod 755*** command on the file. If your user-defined node contains a custom compiler, put your user-defined plug-in file in the same directory.
4. Stop and restart the integration node to implement the change and to ensure that the existence of the new file is detected.

   An integration node restart is not necessary in the following circumstances:

   - If you have created an integration server in the IBM App Connect Enterprise Toolkit, and nothing is yet deployed to it, you can add the `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` file to your selected directory.
   - If something has already been deployed to the integration server that you want to use, add the `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` file to your selected directory, and issue the **mqsireload** command to

restart the group. You cannot overwrite an existing file on the Windows system when the integration node is running, because of the file lock that is put in place by the operating system.

Use these two approaches with care, because any integration server that is connected to the same integration node also detects the new `.lil`, `.pdb`, `.jar`, `.par`, or `.lel` files when that integration server restarts, or when something is first deployed to that integration server.

5. Repeat the previous steps for every integration node that needs the user-defined extension file and user-defined node plug-in file. If all of your integration nodes are on the same operating system type, you can build the user-defined extension file once and distribute it to each of your systems.

   If you have a cluster that includes Linux and Windows integration nodes, you must build the user-defined extension files separately on each operating system type.

   Windows On Windows, the `.pdb` file provides symbolic information that is used when stack diagnostic information is displayed in the event of access violations or other software malfunctions.
6. For C user-defined extensions, store the `.pdb` file in the same directory as the `.lil` file to which it corresponds.
7. Use the **`mqsicreatebroker`** command, to specify to the integration node the directory that contains the user-defined extension file.

   When you have installed a user-defined extension, it is referred to by its schema and name, just like a message flow.

Results

The integration node loads the user-defined extension files during initialization. After loading the files, the integration node calls the registration functions in the user-defined extension and records what nodes or parsers the user-defined extension supports.

A C user-defined extension implements a node or parser factory that can support multiple nodes or parser types. For more information, see "Node and parser factory behavior" on page 2327. Java users are not required to write a node factory.

Installing user-defined extension runtime files on an integration server

Install the compiled runtime files for your user-defined extension on the integration server on which you want to test its function. If your user-defined node uses a custom compiler, install the user-defined node plug-in to the integration server on which you want to deploy the node.

Before you begin

- Create and compile your user-defined extension using the procedure described in "Compiling a Java user-defined node" on page 2418 or "Compiling a C user-defined extension" on page 2400.
  - The files that have been created for extension created in C depend on the operating system:

    Windows A dynamic link library (DLL), named with a file type of `.lil`.

    Linux A shared object, again with a file type of `.lil`.
  - For Java nodes, a Java Archive file (JAR), with a file type of `.jar` (on all operating systems).
- If you have created a user-defined node, you must also complete the task that is described in "Creating the user interface representation of a user-defined node in the IBM App Connect Enterprise Toolkit" on page 2421.
- If your user-defined node contains a custom compiler, you must package your user-defined node project; see "Packaging and distributing a user-defined node project" on page 2452.

About this task

This task instructs you to stop and restart integration servers. This action is required in all cases, although if you do stop and restart an integration server, you should ensure that anyone with an interest in a particular integration server is made aware that recent changes have been made.

This task is applicable to user-defined nodes written in Java or C only. If your user-defined node contains a custom compiler, you must install the compiled runtime files and the user-defined node plug-in to the integration server. If your user-defined node does not contain a custom compiler, install only the compiled runtime files to the integration server.

To install runtime and user-defined node plug-in files on the integration server:

Procedure

1. Stop the integration server on which you want to install your compiled or packaged user-defined extension file (files with extension `.lil`, `.jar`, `.par`, `.pdb`, or `.lel`)
   This is required in all cases, there are no exceptions.
2. Create a directory if you do not already have one for this purpose.

   For example:

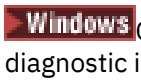
   ```
   C:\Work\Development\Demo\Plugins\bin
   ```

   CAUTION: Do not put the `.lil`, `.jar`, `.par`, `.pdb`, or `.lel` files in the IBM App Connect Enterprise installation directory, because they might be overwritten.
3. Put your user-defined file in the directory, and make sure that the integration server has access to it.

   For example, on Linux use the **chmod 755*** command on the file. If your user-defined node contains a custom compiler, put your user-defined plug-in file in the same directory.
4. Restart the integration server to implement the change and to ensure that the existence of the new file is detected.
5. Repeat the previous steps for every integration server that needs the user-defined extension file and user-defined node plug-in file. If all of your integration servers are on the same operating system type, you can build the user-defined extension file once and distribute it to each of your systems.

   If you have a cluster that includes Linux and Windows integration servers, you must build the user-defined extension files separately on each operating system type.

   On Windows, the `.pdb` file provides symbolic information that is used when stack diagnostic information is displayed in the event of access violations or other software malfunctions.
6. For C user-defined extensions, store the `.pdb` file in the same directory as the `.lil` file to which it corresponds.
7. Add the directory to the LILPATH by updating the `server.conf.yaml` configuration file.

   For example:

   ```
   lilPath: C:\Work\Development\Demo\Plugins\bin
   ```

Results

The integration server loads the user-defined extension files during initialization. After loading the files, the integration server calls the registration functions in the user-defined extension and records what nodes or parsers the user-defined extension supports.

A C user-defined extension implements a node or parser factory that can support multiple nodes or parser types. For more information, see “Node and parser factory behavior” on page 2327. Java users are not required to write a node factory.

Updating a user-defined extension

On all systems, you can change a user-defined extension file.

About this task

You must stop and restart the integration node for your changes to show, However, you are not required to stop and restart the integration node in the following two scenarios:

- If you have created an integration server in the IBM App Connect Enterprise Toolkit, but have not yet deployed to it, you can add the `.lil`, `.pdb`, and `.jar` files to your chosen directory.
- If an object has already been deployed to the integration server that you want to use, add the `.lil`, `.pdb`, and `.jar` files to your chosen directory and use the **mqsireload** command to restart the group. You cannot overwrite an existing file on the Windows system when the integration node is running because a file lock is put in place by the operating system.

These two scenarios must be used with caution, because any integration server that is connected to the same integration node also detects the new `.lil`, `.pdb`, and `.jar` files when that integration server is restarted, or when an object is first deployed to it. If you restart the integration node, you must ensure that anyone with an interest in a particular integration server is made aware that recent changes have been made to the integration node.

These two scenarios assume that you have used the **mqsicreatebroker** command to notify the integration node of the directory in which the user-defined extension files have been placed.

Procedure

To change a user-defined extension file:

1. Stop the integration node by using the **mqsistop** command.
2. Update or overwrite the `.lil` or `.jar` file.
3. Restart the integration node by using the **mqsistart** command.

Uninstalling a user-defined extension from the integration node

Remove a user-defined extension file from the integration node.

About this task

1. Stop the integration node by using the **mqsistop** command.
2. Delete the `.lil` or `.jar` file from the directory you created when installing the user-defined extension. See “Installing user-defined extension runtime files on an integration node” on page 2455.
3. Restart the integration node by using the **mqsistart** command.

Using error logging from a user-defined extension

Program user-defined extensions to write entries in the local error log.

About this task

In most circumstances, user-defined extensions should use exceptions to report errors. However, you can provide information about significant events, error or otherwise, for problem determination and operational purposes. The details that you supply are included in predefined message text that is extracted from a message source or catalog.

- In C code, use the utility function `CciLog` or `CciLogW` to report events. Two of the arguments that you pass to this function, `messageSource` and `messageNumber`, define the event source (catalog) and the integer representation of a message within that source, respectively.

  You can also write trace information, using `CciUserTrace`, `CciUserTraceW`, `CciUserDebugTrace`, and `CciUserDebugTraceW` when tracing and debugging is active.
- In Java code, use the class `MbService`, which provides static methods to log information to the event log. To log messages to the event log, package your messages into a standard Java resource bundle. You can use one of the three logging methods, passing in the resource bundle name and the message key. The message is fully resolved, and is then inserted as a single insert into the appropriate integration node message as shown:
  - logInformation(...) - BIP4360 Java user-defined node information: *user message*
  - logWarning(...) - BIP4361 Java user-defined node warning: *user message*
  - logError(...) - BIP4362 Java user-defined node error: *user message*

You can write messages that are defined in the product message catalog (`BIPmsgs`), to which you can add your own text as an argument. If you prefer, you can create your own message catalog, so that you can create more complex messages, or share a message catalog with other applications. If you want to create your own message catalog, see "Creating message catalogs" on page 2459.

- Windows On Windows systems, messages are written to the Windows event log.
- Linux On Linux systems, messages are written to the SYSLOG facility.

The description here covers exceptions that are raised during normal message flow processing. You must also provide for exceptions that are raised when you deploy and configure a message flow. Messages that result from these configuration exceptions are reported back to the IBM App Connect Enterprise Toolkit for display to the IBM App Connect Enterprise Toolkit user. Create an appropriately-named Java properties file to contain your messages, then copy the file to each computer on which you are running the IBM App Connect Enterprise Toolkit, so that your messages can be displayed.

Creating message catalogs

Create your own message catalogs to write tailored entries to the local error log.

About this task

In some error and other situations, you might choose to write information to the error log so that you can track what is happening in a message flow. You can use the Throw and Trace built-in nodes to generate entries in the log, or you can create your own nodes and user exits, and write entries in the log from your user-defined extensions.

You can write either or both of the following sets of messages:

- A fixed set of messages that are provided in the product message catalog. This set provides a range of numbers for Throw nodes (BIP3001 to BIP3049), and a second range for Trace nodes (BIP3051 to BIP3099). A third range (BIP2951 to BIP2999) is provided for the ESQL statements LOG and THROW.

  When you use these messages, you can also provide additional text that is displayed in the message text.
- Your own messages, created in your own message catalog. You can use this additional catalog to define specialized message content, and you can include variables or inserts that are determined by the code that generates the message. You can also share your own message catalog with other applications that are not associated with IBM App Connect Enterprise.

When you throw an exception from ESQL by using a THROW statement, the ESQL code adds an extra leading insert that contains the name of the current component. The rest of the inserts that are provided by the ESQL script follow this leading insert. Therefore, you must consider this insert when you are writing your own message catalog.

The instructions in this topic describe how to create message catalogs for C programs. If you want to create a Java resource bundle, refer to the documentation for the Java 2 Platform, Standard Edition.

Read the section appropriate to your integration node operating system:

- "Building and installing a Windows message source" on page 2460
- "Creating an XPG/4 catalog for Linux, UNIX, and z/OS" on page 2461

Building and installing a Windows message source

About this task

On Windows, you must create your additional message catalog as a DLL file. The DLL file contains definitions of your event messages, which the event viewer can display in a readable format, based on the event message written by your application. When you compile a message catalog, a header file is created that defines symbolic values for each event message number you have created. You must include the header file in your application.

To create an event source for the Windows Event Log Service:

Procedure

1. Create a message compiler input (`.mc`) file with the source for your event messages.

   Refer to the Microsoft Developer Network Web site, and search on `.mc file` for details on the format of this input file.
2. Compile the message file to create a resource compiler input file:

   ```
   mc -v -w -s -h c:\mymessages -r c:\mymessages mymsg.mc
   ```

   where `c:\mymessages` is the location of the output files and *mymsg.mc* is the name of the input file.

   The message compiler produces an output header (`.h`) file that contains symbolic `#defines` that map to each message number that is coded in the input `.mc` file. Include this header file when you compile a user-defined extension source file that uses a utility function (for example, `CciLog`) to write an event message that you have defined. The `messageNumber` argument to utility function must use the appropriate value that is hash-defined in the output header file.
3. Compile the output file (`.rc`) from the message compiler to create a resource file (`.res`):

   ```
   RC /v output_file.rc
   ```
4. Create a resource DLL file from the `.res` file:

   ```
   LINK /DLL /NOENTRY resource_file.res
   ```
5. Append the location of the resource DLL file to the *MQSI_CONSOLE_NLSPATH* environment variable, for example:

   ```
   set MQSI_CONSOLE_NLSPATH=%MQSI_CONSOLE_NLSPATH%;c:\messages
   ```

   You can do this by creating a custom environment script in your working directory. The default location is `C:\ProgramData\IBM\MQSI\Common\profiles`.

6. Install the event source into the Windows Event Log Service:

   a) Start the Windows Registry Editor:

   ```
   regedit
   ```

   b) Create a new registry subkey for your user-defined extension under the existing structure:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application
```

   Right-click **Application** and select **New** > **Key**. The new key is created immediately under the Application key (not under the IBM App Connect Enterprise key). You must give the key the name that you specify for the `messageSource` on a utility function in your user-defined extension (for example, `CciLog`) or as the property of the built-in node that you have included in your message flow.

   Create the following values for this entry:

   EventMessageFile
   Set the value of this string to contain the fully qualified path for the DLL file that you have created to contain your messages. This entry represents the message catalog.

   TypesSupported
   Set the DWORD value to "7".

Creating an XPG/4 catalog for Linux, UNIX, and z/OS

About this task

On Linux, UNIX, and z/OS systems, messages are written to the SYSLOG facility. If you want to use your own message catalog, you must create an XPG/4 message catalog.

The process for creating a message catalog (a `.cat` file) depends on the operating system on which you are creating it. The commands that you use are typically **gencat** (create or modify a message catalog) and **dspcat** (to display all or part of a message catalog). The **gencat** command merges text files that contain your message text, to create or modify a formatted catalog. The text files typically have a file extension of `.msg`.

You must append the location of the message catalog to the *MQSI_CONSOLE_NLSPATH* environment variable. You can use %L and %N to represent the locale and the catalog name, for example:

```
export MQSI_CONSOLE_NLSPATH=${MQSI_CONSOLE_NLSPATH}:${MY_INST_PATH}/messages/%L/%N:
${MY_INST_PATH}/messages/En_US/%N
```

In this example, the English version is hardcoded later in the search path, ensuring that messages are displayed even in locales for which no `.cat` file exists.

The messages that you define in the `.msg` files can include variables that are substituted at run time. Such variables must be of the format *{number}*, where *{number}* is the message insert number, surrounded by braces. The first message insert is numbered 0. For example:

```
1234 "MSG1234E: \
Syntax Error. \n
The value '{0}' is not valid for property '{1}'.\n
Correct it and then reissue the command.\n"
```

If you create a message catalog on one operating system, you cannot port it to another operating system because the catalogs are binary-encoded. However, you can use the same `.msg` files as input to the **gencat** command on another system.

See the relevant information in the documentation for your operating system. For example:

- AIX For AIX, see the *Commands Reference* in the product documentation.

- z/OS For z/OS, see the *UNIX System Services Command Reference* in the z/OS product documentation online.

You must also check the information about additional supported locales, if you want to use messages in locales other than US English.

Chapter 7. Deploying integration solutions

Deployment is the process of transferring the development resources for an integration solution to the runtime environment. The development resources are deployed to an integration server or to the cloud. To deploy an integration solution to a production environment, you can package your development resources into a single file for transfer to the production environment.

About this task

To develop an integration solution, you might need to create a number of resources such as message flows, subflows, message models, maps, Java code, and ESQL files.

You can store your development resources in applications or libraries. If some of your resources are used by multiple applications, you can store those resources in a static library or a shared library. To understand the difference between static libraries and shared libraries, see “Libraries” on page 1945. You can then deploy the resources by deploying the applications and libraries that contain the resources.

During the development of a solution, you can deploy the development resources directly to an integration server by using options in the IBM App Connect Enterprise Toolkit. By using this approach, you can quickly verify changes as you develop your solution; see “Deploying integration solutions during development” on page 2464.

When you are ready to deploy your solution to a production environment, you can package the resources into a BAR file. You can add applications and libraries to the BAR file. If your solution contains message flows or other resources that are in an integration project, you can add these resources individually to the BAR file. If you use policies to override the properties that are set on the message flows and message flow nodes, you also add these policies to the BAR file.

You can also include version information in your resources so that you can match the packaged or deployed resources with the resources that you maintain in a source control system.

For more information, see “Packaging integration solutions” on page 2465.

Before you deploy your solution to a production environment, you can customize the BAR file to configure the solution for any differences between the development environment and the production environment.

You can also choose to precompile the resources that the BAR file contains (such as XML schema, DFDL schema, and graphical data maps), so that they are ready to run when the integration server starts. This reduces the time that it takes for the deployed application to start processing messages. For more information, see the command.

When your BAR file is configured, you can deploy it to a production environment; see “Deploying integration solutions to a production environment” on page 2480. Your production environment could be IBM App Connect Enterprise or IBM App Connect on IBM Cloud.

When an integration solution is no longer required, you can remove the integration solution from your environment; see “Deleting deployed resources” on page 319. If you want to redeploy your solution, you must first delete the existing deployed solution and then deploy again.

You can also deploy IBM App Connect Enterprise to IBM Cloud Private, by using Helm Charts. For more information, see IBM App Connect in containers.

Deploying integration solutions during development

During development, you can deploy a solution directly to an integration server by using options in the IBM App Connect Enterprise Toolkit.

Before you begin

You must have an integration server in the IBM App Connect Enterprise Toolkit that you can deploy resources to. For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

About this task

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers and message flows that you can deploy. For more information, see "Restrictions that apply in each operation mode" on page 5.

If you are developing a solution and you want to test the solution or some of the solution resources quickly, you can deploy the resources that you want to test directly to an integration server. You can deploy the following resources:

- An application or library
- A message flow that is contained in an integration project
- A subflow that is defined in a `.subflow` file and contained in an integration project
- One or more policies that are contained in a policy project

For prerequisites and guidelines that are associated with the deployment of integration solutions, see "Deployment rules and guidelines" on page 2465.

Procedure

In the IBM App Connect Enterprise Toolkit, deploy your resources by using one of the following methods:

- Use drag-and-drop:
  a) In the **Application Development** pane, select the resources that you want to deploy.
  b) Drag the resources onto an integration server in the **Integration Explorer** pane.
- Use menu options that are associated with the resource:
  a) In the **Application Development** pane, right-click the resource that you want to deploy and click **Deploy**.
  b) In the **Integration Explorer** pane, select the integration server where you want to deploy the resource and click **Finish**.

  Repeat these steps to deploy any other resources that you want to deploy.
- Use menu options that are associated with the integration server:
  a) In the **Integration Explorer** pane, right-click the integration server where you want to deploy your resources, and click **Deploy**.
  b) Select the type of resource that you want to deploy to see a list of available resources of that type that can be deployed.
  c) Select the resource that you want to deploy and click **OK**.

  Repeat these steps to deploy any other resources that you want to deploy.

Results

Your solution resources are deployed. One or more BAR files are generated automatically, and deployed to the integration server. You can find these BAR files in the **GeneratedBarFiles** folder in the Application Development view. If you deploy an application that refers to a static library, the static library is added to

the same BAR file as the application. If you deploy an application that refers to a shared library, you must first deploy the shared library; a separate BAR file is generated for the shared library.

Deployment rules and guidelines

When you deploy all or part of an integration solution to an integration server, you must adhere to a number of rules.

Check the following list for any rules or guidelines for deploying all or part of your integration solution:

- If you are deploying an application that depends on a shared library, you must deploy the shared library with the application or before you deploy the application.
- If you deploy a message flow that is contained in an application or static library, the whole application or library is deployed; you cannot deploy a message flow on its own if the message flow belongs to an application or static library. You cannot store a message flow in a shared library.
- If you deploy a message flow that contains a subflow that is defined in a `.subflow` file, you must deploy the subflow to the same integration server. You must deploy the subflow with the message flow or before you deploy the message flow.
- If you redeploy a subflow that is defined in a `.subflow` file to an integration server, any message flows that use the subflow in that integration server are stopped and restarted. When the message flows restart, they use the updated subflow.
- If you deploy an application or library to an integration server where the application or library already exists, the existing application or library and its contents are removed, before the new application or library is deployed.
- If you are deploying a message flow that uses one or more Mapping nodes, see Deploying message maps.
- If you are deploying a message flow that uses WebSphere Adapters, see “WebSphere Adapters resources” on page 2475.
- Windows If your message flows call .NET code from a .NETCompute node or from ESQL, you must distribute the assemblies to every integration server that uses these message flows. For more information, see “Deploying .NET assemblies” on page 1793.
- If your BAR file contains a mixture of resources that are compiled and resources that are not compiled, you might see unexpected results. For example, if you select the **Compile and in-line resources** option to create a BAR file that contains an ESQL file and a message flow, the ESQL is embedded in the compiled message flow (`.cmf`) file. If you then update the ESQL and add it to the BAR file with the **Compile and in-line resources** option cleared, the ESQL file is added as an individual resource, but the `.cmf` file uses the original ESQL because the original ESQL remains embedded in the `.cmf` file. To avoid confusion, select one of the following deployment options:

  1. Deploy all the resources (for example `.msgflow`, `.subflow`, and `.esql`) as source code.
  2. Deploy all the resources in the compiled form. This deployment method requires that all the subflows are implemented in `.msgflow` files. As a result, all the flow, subflow, and ESQL code is in-lined into the parent CMF file. This method does not allow the same flexibility as the source code deployment, but it does provide an unambiguous runtime behavior.

  For other limitations when using CMF files, see BAR file contents: Message flows.

Packaging integration solutions

If you package your integration solution, you can control the deployment of the solution into your production environment. You package integration solutions by adding the required resources to a BAR file.

Before you begin

Develop and test the integration solutions that you want to deploy; see Chapter 6, “Developing integration solutions,” on page 481 and “Testing and troubleshooting message flows” on page 647.

About this task

You can deploy some individual resources without packaging them manually; see “Deploying integration solutions during development” on page 2464.

However, you can deploy all your integration solution resources together if you package them in a single BAR file, then deploy the BAR file.

You can also include version information in your resources so that you can match the packaged or deployed resources with the resources that you maintain in a source control system; see “Version and keyword information for deployable resources” on page 2466.

Procedure

To package resources into a BAR file, complete the following steps:

1. Create a BAR file; see “Creating a BAR file” on page 2469.
2. Add resources to the BAR file; see “Adding resources to a BAR file” on page 2470.

What to do next

1. Optional: Prepare your packaged solution for deployment by configuring properties with the appropriate values for your production environment.
2. Deploy your BAR file to an integration server, to IBM Integration Bus on Cloud, or to IBM App Connect on IBM Cloud; see “Deploying integration solutions to a production environment” on page 2480.

Version and keyword information for deployable resources

You can add a version and other custom keyword values to deployable resources during development and then check the versions of resources that are packaged in BAR files, or deployed to an integration server.

The ability to add version information to your resources is particularly useful if you use a source control system to manage your resources. The version information that you add to your resources can be viewed when the resources are packaged into a BAR file ready for deployment, and when the resources are deployed to an integration server. You can use the version information to match the versions of the packaged or deployed resources with the versions of the resources that you store in your source control system.

For information about adding and viewing version and keyword information in your deployable resources, see the following topics:

- “Adding version information and custom keyword values to your development resources” on page 2466
- “Viewing version information and custom keyword values in your packaged solutions” on page 2468
- “Viewing version information and custom keyword values in your deployed solutions” on page 319

Adding version information and custom keyword values to your development resources

You can add a version and other custom keyword values to deployable resources during development.

About this task

Version information is assigned to a development resource by including the following string in the source code of the resource:

```
$MQSI_VERSION=VersionIdentifier MQSI$
```

For example:

```
$MQSI_VERSION=v1.0 MQSI$
```

Custom keywords are assigned to a development resource by including the following string in the source code of the resource:

```
$MQSI KeywordName=KeywordValue MQSI$
```

For example:

```
$MQSI Author=John Smith MQSI$
```

Note: Do not use BAR as a keyword name. The BAR keyword is associated with each object automatically when it is deployed and it contains the full path name of the BAR file that deployed the object.

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

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

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

- `$MQSI RCSVER=$id$ MQSI$` is acceptable (`RCSVER` is a valid keyword name).
- `$MQSI $name=Fred MQSI$` is not acceptable (`$name` is not a valid keyword name).

The method for adding version information, custom keywords, and custom keyword values to the source code varies depending on the resource type. The following table describes a method that you can use to add version and custom keyword information to each type of resource:

Resource type	Method to add version and custom keywords
Message flows	Add the *VersionIdentifier* to the Version field in the message flow properties. Information added to the Version field does not need to be encased in `$MQSI` tags. Add custom keyword strings to the Short Description or Long Description fields in the message flow properties by using the IBM App Connect Enterprise Toolkit or the web user interface.
Subflows	Add the *VersionIdentifier* to the Version field in the subflow properties. Information added to the Version field does not need to be encased in `$MQSI` tags. Add custom keyword strings to the Short Description or Long Description fields in the subflow properties by using the IBM App Connect Enterprise Toolkit or the web user interface. **Note:** If you use `Compile and in-line resources` when you build your BAR file, you must use a different keyword in each instance of a subflow. Only the first recorded instance of each keyword within the message flow `.cmf` file is available to applications that use the IBM Integration API, which includes the IBM App Connect Enterprise Toolkit. The order that subflows appear in the `.cmf` file is not guaranteed.
Applications and Libraries	It is not possible to add version or custom keyword strings to an application or library during development. Instead, you must complete the following steps: 1. Create a `keywords.txt` file. 2. Add the version or custom keyword strings to the `keywords.txt` file. 3. Package the application or library in a BAR file. 4. Add the `keywords.txt` file to the `META-INF` directory within the `.appzip`, `.libzip`, or `.shlibzip` file that is in the BAR file.

For information about viewing version or custom keyword information in your packaged or deployed resources, see the following topics:

- "Viewing version information and custom keyword values in your packaged solutions" on page 2468
- "Viewing version information and custom keyword values in your deployed solutions" on page 319

Viewing version information and custom keyword values in your packaged solutions

You can view version and custom keyword information in your packaged resources.

About this task

If version or custom keyword information was added to your resources during development, you can view the values in a BAR file by using the following methods:

- "Viewing version information by using the BAR file editor" on page 2468
- "Viewing version and custom keyword information by using the mqsireadbar command" on page 2468

Viewing version information by using the BAR file editor

Procedure

To view the version that is associated with the resources that are packaged in a BAR file by using the BAR file editor, complete the following steps:

1. Open the BAR file in the IBM App Connect Enterprise Toolkit.

   The BAR file opens in the BAR file editor.
2. Expand the entries in the **Name** column to see all the resources in the BAR file.

Results

If any of the resources have a version, the value is displayed in the **Version** column. It is not possible to view custom keywords and values by using the BAR file editor.

Viewing version and custom keyword information by using the `mqsireadbar` command

Procedure

From the command environment (on Linux) or the IBM App Connect Enterprise Console (on Windows), type the following command:

```
mqsireadbar -b BARFileName -r
```

where *BARFileName* is the fully qualified name of your BAR file.

Results

The content of the BAR file is listed. If a resource has a version or custom keyword value, the values are listed under the resource name. For example, in the following system output, you can see that the ESQL resource has an associated version (`esql2.0`) and custom keyword and value (`Author = John Smith`).

```
BIP1052I: Reading Bar file using runtime mqsireadbar...
Error Handler Message Flowsproject.generated.bar:
  Error Handler Message Flows.appzip (12/06/15 17:46):
    Error_Handler.esql (12/06/15 17:46):
    application.descriptor (12/06/15 17:46):
    Main_Flow.msgflow (12/06/15 17:46):
    Error_Handler.subflow (12/06/15 17:46):
    Main_Flow.esql (12/06/15 17:46):
```

```
      Author = John Smith
      VERSION = esql2.0
    Deployment descriptor:
      startMode
      javaIsolation
      Error_Handler#Check Backout Count.dataSource
      Error_Handler#Check Backout Count.connectDatasourceBeforeFlowStarts
      Error_Handler#Copy Message.dataSource
...
```

What to do next

For information about adding version or custom keyword information to your development resources, or viewing version or custom keyword information in your deployed resources, see the following topics:

- "Adding version information and custom keyword values to your development resources" on page 2466
- "Viewing version information and custom keyword values in your deployed solutions" on page 319

Creating a BAR file

Create a separate BAR file for each configuration that you want to deploy.

About this task

When you are ready to deploy your solution to a production environment, you can package the resources into a BAR file. You can add applications and libraries to the BAR file. If your solution contains message flows or other resources that are in an integration project, you can add these resources individually. If you have policies that can be used to override message flow and message flow node properties at run time, you also add them to the BAR file. When you have added all the resources that are required by your solution to your bar file, you can deploy the BAR file to an integration server.

The following instructions describe how to create a BAR file.

Creating a BAR file with the IBM App Connect Enterprise Toolkit

About this task

When you create a BAR file by using the IBM App Connect Enterprise Toolkit, by default it is created in an integration project called **BARfiles**. In the Application Development view, the new BAR file is shown under the **BARfiles** integration project, in a folder called BARs. You can customize the default location of BAR files by setting a preference (**Window** > **Preferences** > **Integration Development** > **Build BAR**).

To create a BAR file by using the IBM App Connect Enterprise Toolkit, complete the following steps:

Procedure

1. Click **File** > **New** > **BAR file**.
2. Select the location for your BAR file.

   You can select an existing application, library, or integration project, or you can create a new integration project by clicking **New**.
3. Enter a name for the BAR file that you are creating.
4. Click **Finish**.

Results

A file with a `.bar` extension is created and is displayed under the application, library, or integration project, in a folder called BARs. The BAR File editor opens. All BAR files are also shown in a separate category in the Application Development view called BARs. Each BAR file that is listed here displays information about the project in which the BAR file is stored.

What to do next

Add files to the BAR file by following the instructions in "Adding resources to a BAR file" on page 2470.

Adding resources to a BAR file

To deploy a group of related resources to an integration server, include them in a BAR file.

Before you begin

Create a BAR file for each configuration that you want to deploy. For more information, see "Creating a BAR file" on page 2469.

About this task

You can add any deployable resources from the workspace to a BAR file. If you select **Add workspace project source files**, the source and project files for all message flows, message model schema files, message sets, or other deployable resources in the BAR file are included. If you do not add the source and project files to the BAR file, make sure that you keep a copy of the source and project files because you cannot regenerate the development resources from the compiled resources that are in the BAR file. If you add the source and project files to the BAR file, make sure that you keep a copy of the BAR file because you cannot regenerate the BAR file or from the deployed resources on an integration server.

For more information about the files that you can include in a BAR file, see "BAR file contents" on page 2472.

To add files to a BAR file by using the IBM App Connect Enterprise Toolkit, complete the following steps.

Procedure

1. Open the BAR file by double-clicking it.

   The contents of the BAR file are shown in the BAR File editor. (If the BAR file is new, this view is empty.)
2. On the **Prepare** tab of the BAR File editor, select deployable workspace resources to add to the BAR file:

   - Click **Applications, shared libraries, services, and REST APIs** to see a list of these resources that can be deployed. If you select an application, shared library, or service, all the contained resources are deployed with that container.
   - Click **Policies** to see a list of policy projects that can be deployed. You can add multiple policy projects to the BAR file. Properties that are set in these policies override the properties that are set on deployed message flows, message flow nodes, and the integration server at run time.
   - Click **Message flows, static libraries, and other message flow dependencies** to see the individual resources that you can deploy directly to the integration server. After these resources are deployed, they are visible to all other deployed resources in the same integration server.

   Note:

   - If an application refers to a shared library, that library must be deployed before or with the application. A shared library can be deployed in the same BAR file as an application that refers to it, or the application and referenced shared library can be deployed in separate BAR files.
   - If an application that you select refers to one or more static libraries, those libraries are added to the application `.appzip` file as nested `.libzip` files. These referenced static libraries are private and are not accessible to resources outside the application. Therefore, those static libraries are not selected on the **Prepare** tab by default. However, if you want to deploy a static library as an integration server-level library, which is accessible to other resources, select the static library to

deploy it separately. A `.libzip` file is created for that static library and added to the BAR file at root level.

- If you select a static library that refers to other static libraries, those referenced libraries are also added to the BAR file as `.libzip` files.
- If the application that you select refers to one or more .NET application domains, those application domains are added to the BAR file at root level as a `.appdomainzip` file.
- You cannot add an integration project to a BAR file. Instead, add the individual resources that are contained in the integration project to the BAR file.
- If your solution contains Java code or message sets, you must compile these resources and add the compiled resources to the BAR file; see "Packaging resources that include Java code or message sets" on page 2474.
- If a message flow that you select contains one or more WebSphere Adapters nodes, a dialog box opens so that you can identify the following resources:
  - One or more WebSphere Adapters components to be used by the WebSphere Adapters nodes.
  - One or more libraries that contain an XSD for the business objects that are used by the WebSphere Adapters nodes

  For more information about including WebSphere Adapters in a BAR file, see "Including WebSphere Adapters resources in a BAR file" on page 2475.

3. Optional: To include source files, select **Add workspace project source files**.

   If you select **Add workspace project source files**, source projects for all applications, libraries, and other compiled resources are added to the `src` folder of the BAR file.
4. Optional: To remove existing content from the BAR file before you build the new BAR file, select **Remove contents of BAR before building**.
5. Optional: If you are adding a message flow to a BAR for a second time, and used the **Manage** tab to change flow parameters, select **Override configurable property values** to reset configuration settings.

   If this control is cleared, existing settings are left in place when a flow is replaced.
6. Optional: To include message flows as compiled message flow (`.cmf`) files, and to include ESQL code directly in the `.cmf` file of each message flow that references an ESQL file, select **Compile and in-line resources**. By default, when you add a message flow to a BAR file, it is added as a `.msgflow` file. By default, each ESQL file that is referenced by one or more of your message flows is deployed as an individual resource, and can be accessed by multiple `.msgflow` files.

   If any of the flows that you add to your BAR file contain a subflow that is defined in a `.msgflow` file, you must select **Compile and in-line resources**.
7. Click **Build and Save**.

Results

The **Manage** tab lists the files that are now in your BAR file. Expand your applications and libraries to view their contents. Compiled resources in the BAR file are shown in alphabetical order in the tree. You can edit only root elements. For example, if you added an application to the BAR file, you can rename the `.appzip` file or add comments to it. You cannot edit the resources inside the `.appzip` file, but you can edit the configurable properties for those resources.

You cannot remove individual resources from `.appzip`, `.appdomainzip`, `.libzip`, or `.shlibzip` files. To remove these resources, you must remove the application, .NET application domain, or library.

You can choose not to display your source files by selecting **Built resources** or **Configurable properties** from the list in the **Filter by** menu.

What to do next

Complete one or both of the following tasks:

- Deploy the BAR file; see "Deploying integration solutions to a production environment" on page 2480.

BAR file contents

The BAR file is a compressed file to which you can add a number of deployable resources.

When you select resources to add to the BAR file, the following files are added for each resource type:

Applications

A `.appzip` file for each application.

The `.appzip` file contains all resources that belong to the application, such as `.msgflow`, `.cmf`, `.esql`, `.map`, `.xsd`, and any message set `.dictionary` and `.xsdzip` files. If an application refers to one or more libraries, `.libzip` or `.shlibzip` files for the referenced libraries are also added to the BAR file.

Static libraries

A `.libzip` file for each static library.

This file contains all resources that belong to the static library, such as `.msgflow`, `.cmf`, `.esql`, `.map`, `.xsd`, and any message set `.dictionary` and `.xsdzip` files. If a static library refers to other static libraries, `.libzip` files for the referenced libraries are also added to the BAR file.

Shared libraries

A `.shlibzip` file for each shared library.

This file contains all resources that belong to the shared library, such as `.cmf`, `.esql`, `.map`, `.xsd`, and any message set `.dictionary` and `.xsdzip` files. If a shared library refers to other shared libraries, `.shlibzip` files for the referenced libraries are also added to the BAR file.

Message flows

- If you select **Compile and in-line resources** in the BAR File editor, a `.cmf` file is added to the BAR file for each message flow. This file is a compiled version of the message flow.

  The following limitations apply when you are testing compiled message flows:

  – You cannot use the Flow Exerciser with compiled message flows.

  – You cannot set debugger breakpoints on connections in compiled message flows.

  – The flow diagram that is shown in the web user interface does not match how the flow is shown in the IBM App Connect Enterprise Toolkit.

  – Monitoring events and statistics records do not match the points in the original flow diagram.

- If you do not select **Compile and in-line resources** in the BAR File editor, a `.msgflow` file is added to the BAR file for each message flow. This file contains a definition for the message flow. This file is not compiled.

You cannot add the same message flow to a BAR file as both a `.cmf` file and a `.msgflow` file. If your flow contains one of the following nodes, you cannot add the flow as a `.msgflow` file:

- A user-defined node that is created from a subflow
- A subflow node that represents a subflow that is defined in a `.msgflow` file
- A WebSphere Message Broker Version 7.0 Mapping node

Subflows

A `.subflow` file for each subflow that is contained in an integration project. This file is not compiled.

Subflows that are defined in `.msgflow` files are not displayed in the BAR file as separate items, and are added to the BAR file automatically when the BAR file is built. To include these subflows, you must add only the parent flow. Subflows that are defined in `.subflow` files are displayed in the BAR file as separate items and can be deployed as individual resources.

.NET assemblies

A `.appdomainzip` file for each AppDomain that contains .NET assemblies that are used by a message flow in the BAR file.

.NET assemblies that are required by .NETCompute nodes in message flows are added to the BAR file from your integration project automatically when you add the message flow.

Message set dictionaries
A `.dictionary` file for each message set dictionary.

DFDL, XML schema, or WSDL files
The following files are added:

- A `.xsd` file for each DFDL or XML schema file that is defined in an application or library.
- XSD compressed files (`.xsdzip`) for each XML schema and WSDL that is defined in a message set.

XML files, stylesheets, and XSLT files
XML files (`.xml`), stylesheets (`.xsl`), and XSLT files (`.xslt`), if required by nodes in the message flows that you added to this BAR file. For example, the XSLTransform node might require these files.

XML and XSL files are added to the BAR file automatically if they are required by the flow

JAR files
JAR files (`.jar`), if required by JavaCompute nodes in the message flows that you added to this BAR file.

JAR files that are required by JavaCompute nodes in message flows are added to the BAR file from your Java project or integration project automatically when you add the message flow.

JAR files that are available to the integration server include JAR files that you deployed and JAR files that exist in the `classes` directories in the installation directory. For example, the files `IntegrationAPI.jar`, `jplugin2.jar`, and `javacompute.jar` are always visible to the integration server, and do not have to be deployed separately.

WebSphere Adapter files
Inbound or outbound adapter files (`.inadapter` or `.outadapter`), if required by WebSphere Adapter nodes (for example, the SiebelInput node) in the message flows that you added to this BAR file.

ESQL files

- If you select **Compile and in-line resources** in the BAR File editor, ESQL code is embedded in the `.cmf` file that references it.
- If you do not select **Compile and in-line resources** in the BAR File editor, ESQL files (`.esql`) are added to the BAR file as individual resources.

If your BAR file contains a mixture of resources that are compiled and resources that are not compiled, you might see unexpected results. For example, if you select the **Compile and in-line resources** option to create a BAR file that contains an ESQL file and a message flow, the ESQL is embedded in the compiled message flow (`.cmf`) file. If you then update the ESQL and add it to the BAR file with the **Compile and in-line resources** option cleared, the ESQL file is added as an individual resource, but the `.cmf` file uses the original ESQL because the original ESQL remains embedded in the `.cmf` file. To ensure that all your resources are either compiled or not compiled, when you change the **Compile and in-line resources** option, also select **Remove contents of BAR before building**, and rebuild the BAR file.

Graphical maps
A `.map` file for each graphical data mapping routine.

Policies
A `.policyxml` file for each policy.

The `.policyxml` file contains a list of property values that override, at run time, properties that are set on deployed message flows, message flow nodes, or the integration server.

Other files that you might want to associate with this BAR file
You might want to include Java source files, `.msgflow` files, or `.wsdl` files for future reference. BAR files can contain all files types. If you choose to include source files in the BAR file, source projects for all applications, libraries, and other compiled resources are added to the `src` folder of the BAR file.

Packaging resources that include Java code or message sets

Compile Java code and message sets so that you can add them to a BAR file.

About this task

You can use the **mqsipackagebar** command to create BAR files on computers that do not have IBM App Connect Enterprise Toolkit installed. This command adds only deployable resources to the BAR file, so if you want to add message sets or Java code you must first compile them to create deployable resources. For more information about the **mqsipackagebar** command, see command.

You can compile resources by using the **mqsicreatebar** command, or by using the IBM App Connect Enterprise Toolkit. If you use the IBM App Connect Enterprise Toolkit, you can either compile resources in the current workspace or you can compile resources in selected projects.

If you want to use the **mqsipackagebar** command to create BAR files from source files that are stored in a version control system, use one of the following processes to ensure that your message sets and Java code are compiled before you create a BAR file:

- Use the **mqsicreatebar** command to compile message sets and Java code that are stored in the version control system. You might use this command as part of an automated process.
- Ensure that your message sets and Java code are compiled before you add them to the version control system by using the IBM App Connect Enterprise Toolkit.

Procedure

To compile your message sets and Java code, complete one of the following steps:

- To compile message sets and Java code by using the **mqsicreatebar** command, complete the following steps. For more information about the **mqsicreatebar** command, see command.
  a) Open a command window that is configured for your environment.
  b) Enter the following command:

  ```
  mqsicreatebar -data workspace -compileOnly
  ```

  where *workspace* is the fully qualified path to the workspace that contains the message set projects or Java projects that you want to compile.

  Your message sets are compiled into `.dictionary`, or `.xsdzip` files. Your Java code is compiled into `.jar` files.
- To compile all message sets and Java code in the current workspace by using the IBM App Connect Enterprise Toolkit, complete the following steps:
  a) Click **Project** > **Build for mqsipackagebar**.

  The **Build Information** window shows the directories in which the compiled resources are created. Your message sets are compiled into `.dictionary`, or `.xsdzip` files. Your Java code is compiled into `.jar` files.
  b) Click **OK**.
- To compile message sets and Java code in selected projects by using the IBM App Connect Enterprise Toolkit, complete the following steps:
  a) In the Application Development view, right-click the message set project or Java project that you want to compile. Click **Build for mqsipackagebar**.

  The **Build Information** window shows the directories in which the compiled resources are created. Your message sets are compiled into `.dictionary`, or `.xsdzip` files. Your Java code is compiled into `.jar` files.
  b) Click **OK**.

What to do next

Run the **mqsipackagebar** command to create your BAR file, see "Creating a BAR file" on page 2469.

Including WebSphere Adapters resources in a BAR file

Deploy the resources that are generated when you run the **Adapter Connection** wizard by adding them to a BAR file.

Before you begin

- Read "WebSphere Adapters nodes" on page 1050.

About this task

To deploy the message flow successfully, you must deploy the WebSphere Adapters component in the same application, static library, or BAR file as your message flow. If the WebSphere Adapters component is not available, deployment of the message flow fails. The following list includes the file extensions of the resources that you deploy:

- `.msgflow` (the message flow)
- `.inadapter` (the inbound WebSphere Adapters component)
- `.outadapter` (the outbound WebSphere Adapters component)
- Optional: `.appzip` (an application)
- Optional: `.libzip` (a static library)
- Optional: `.shlibzip` (a shared library)

For more information, see "WebSphere Adapters resources" on page 2475.

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers and message flows that you can deploy. For more information, see "Restrictions that apply in each operation mode" on page 5.

Procedure

1. For details of the steps that you must complete before you can deploy a message flow, see "Deployment rules and guidelines" on page 2465.
2. Add the message flow to the BAR file.

   When you add a message flow that contains one or more WebSphere Adapters nodes to a BAR file, a dialog box opens so that you can identify the following resources:

   - One or more WebSphere Adapters components to be used by the WebSphere Adapters nodes
   - One or more libraries that contain an XSD for the business objects that are used by the WebSphere Adapters nodes
3. When you have added the message flow, WebSphere Adapters components, and library, deploy the BAR file.

WebSphere Adapters resources

After you run the **Adapter Connection** wizard and create a message flow, you must deploy the resources that are generated by adding them to an application, library, or BAR file.

To build an IBM App Connect Enterprise solution that integrates with an Enterprise Information System (EIS), such as SAP, Siebel, JD Edwards, or PeopleSoft, you need the following resources:

- A message flow that contains at least one WebSphere Adapters node
- A library (created by the **Adapter Connection** wizard) that describes the logical model of the data, as defined by the EIS

- An adapter, which includes two sets of information that are used by the WebSphere Adapters node in the message flow:
  - Connection information
  - Interface information

    The interface information contains a list of methods. For outbound adapters, methods define the operations or services that can be run on the EIS by the WebSphere Adapters request node. For inbound adapters, the method defines callout functions or events from the EIS that cause the WebSphere Adapters input node to propagate a message through the message flow.

    For each method, the information consists of the name of the method, and the name and namespaces of the message types that are used for input and output. To run the method successfully, the message types must be defined in the library.

Iterative deployment

The message flow can be coded with knowledge of the logical model of the data that is exchanged with the EIS (for example, where Mapping nodes are used to transform the data), but it can also act as a gateway to the EIS, where no transformation of data takes place. When the flow acts as a gateway, you need to be able to run new operations or respond to new events in the EIS without changing or reloading the resources that are already deployed.

You can use iterative deployment to deploy the resources that are required to support the new methods, without affecting any resources that are already deployed. Iterative deployment is possible by using primary and secondary adapters. The primary adapter for a WebSphere Adapters node contains its connection information and part of its interface; the secondary adapters contain the rest of the interface.

You can store a secondary adapter in the same library as the primary adapter, or a different library. If the library that contains the primary adapter is referenced by an application or library, you can reference the project that contains the secondary adapter from the same application or library, or from a different application or library.

You must stop any message flow that references a primary adapter before you redeploy the primary adapter. However, you do not need to stop the message flow when you deploy the secondary adapter. During deployment of the secondary adapter, the primary adapter provides connection information, and the secondary adapter supplies additional interface information.

Preparing packaged solutions for deployment

You can configure properties on the resources in your BAR file to match the properties or requirements of your production environment.

About this task

You can configure the following properties that are associated with your message flows:

- The name of the consumer and provider policies that are used to authenticate, encrypt, and sign messages for SOAP nodes.
- Whether the message flow is processed as an XA coordinated transaction that is coordinated by IBM MQ.
- The name of the security profile that a message flow uses to complete authorization with Security Profiles policy (SecurityProfiles)
- The Workload Management policy that is used to manage the message flow. If you do not have a Workload Management policy, you can set the workload management properties directly in the BAR file.

For information on how to set these properties; see “Editing configurable properties in a BAR file” on page 2477.

You can configure the value of user-defined properties that were configured on your message flows during the development of your solution; see “Configuring a message flow at deployment time with user-defined properties” on page 1749.

You can add additional instances of a message flow to your solution; see “Adding multiple instances of a message flow to a BAR file” on page 2479.

Editing configurable properties in a BAR file

An administrator can use configurable properties to update target-dependent properties that are associated with a solution, such as queue names, queue manager names, and database connections.

Before you begin

Create a BAR file and add resources to it. For more information, see “Creating a BAR file” on page 2469 and “Adding resources to a BAR file” on page 2470.

About this task

By changing configurable properties, you can customize a BAR file for a new integration node (for example, a test system) without editing and rebuilding the message flows or the resources that they use, such as message mappings, ESQL code, and Java code. For example, if you select an application, you can set the start mode for that application. If you select a message flow, you can change properties for policy sets, monitoring, and the security profile. You can also expand a message flow and set the properties for the nodes in that flow. For information about the configurable properties that you can set, see Configurable properties in a file.

Properties that you define are contained in the deployment descriptor file (`META-INF/broker.xml`). The deployment descriptor is parsed when the BAR file is deployed.

You can edit configurable properties in the following ways:

- “Editing configurable properties with the IBM App Connect Enterprise Toolkit” on page 2477
- “Editing configurable properties with the mqsiapplybaroverride command” on page 2478

Editing configurable properties with the IBM App Connect Enterprise Toolkit

About this task

To edit properties by using the IBM App Connect Enterprise Toolkit, complete the following steps.

Procedure

1. Open the BAR file.

   The resources in your BAR file are listed on the **Manage** tab.
2. Optional: You can view the properties that can be configured for your message flows and subflows by selecting **Built resources that are configurable** from the **Filter by** list.
3. Optional: If you migrated the BAR file from a previous version, refresh the contents so that you can see and configure all the available properties. For more information, see “Refreshing the contents of a BAR file” on page 2488.
4. Select the resource that you want to configure.

   You can expand resources to show individual message flow nodes.

   The values that you can configure for the selected resource are displayed in the **Properties** view, on the **Configure** and **Workload Management** tabs.
5. Update the appropriate values in the **Properties** view; see Configurable properties in a file.
6. Save the BAR file.

What to do next

Deploy the BAR file by following the instructions in “Deploying integration solutions to a production environment” on page 2480.

Editing configurable properties with the `mqsiapplybaroverride` command

About this task

To edit properties by using the **mqsiapplybaroverride** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.
2. Run the **mqsireadbar** command to see which properties you can configure; see command.
   For example, the following command reads the file `c:\myBarFile.bar` and lists the properties for all applications and libraries in the file, including libraries inside applications:

   ```
   mqsireadbar -b c:\myBarFile.bar -r
   ```

3. Create a text file, and list the properties that you want to change in the following format:

   ```
   property=newValue
   ```

   For example, the following text file sets the value of the `queueName` property on the node `MQ Input` in the message flow `sampleFlow`, to `NEW_INPUT_QUEUE`, and clears the value of the `queueName` property on the subflow `sampleSubflow1`:

   ```
   # override.properties file
   sampleFlow#MQ Input.queueName=NEW_INPUT_QUEUE
   sampleSubflow1#queueName
   ```

4. Run the **mqsiapplybaroverride** command, and specify the location of your BAR file, and the text file that lists the properties to change.
   For example, the following command reads the BAR file `c:\myBarFile.bar`, uses the `c:\override.properties` file to change the property values set in `c:\myBarFile.bar` and creates a new BAR file, `c:\myNewBarFile.bar`, with the resources you configured to use the new properties:

   ```
   mqsiapplybaroverride -b c:\myBarFile.bar -p c:\override.properties -o c:\myNewBarFile.bar
   ```

   For more information about the **mqsiapplybaroverride** command, see command.

What to do next

Deploy the BAR file by following the instructions in “Deploying integration solutions to a production environment” on page 2480.

Configuring a message flow at deployment time with user-defined properties

Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

Before you begin

For an overview of user-defined properties, see “User-defined properties” on page 569.

For an example of how to code a UDP statement, see DECLARE statement.

About this task

In ESQL, you can define UDPs at the module or schema level. After a UDP has been defined in the Message Flow editor, you can modify the value before you deploy it. You can also modify the value of a

UDP dynamically at run time by using the administration REST API, as described in "Setting message flow user-defined properties at run time by using the administration REST API" on page 436.

To configure UDPs, complete the following steps.

Procedure

1. Open the BAR file.

   The contents of the BAR file are shown on the **Manage** tab of the BAR File editor. On this tab, you can expand a flow to show the individual nodes that it contains.
2. Click the relevant message flow or subflow (not the `.cmf` compiled message flow file).

   The UDPs that are defined in that flow are displayed with their values in the **Properties** view.
3. If the value of the UDP is unsuitable for your current environment or task, change it to an appropriate value.

   The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow.

   Note: If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
4. Save your BAR file.

What to do next

Deploy the message flow by following the instructions in "Deploying integration solutions to a production environment" on page 2480.

Adding multiple instances of a message flow to a BAR file

Add more than one instance of a message flow to a BAR file and use configurable properties to customize each instance.

About this task

You can deploy multiple instances of the same message flow with different values for configurable properties. By using the same flow with different configurable property values, you can reuse a flow but customize each instance to your requirements. For example, you might use one flow with a particular value for an output queue, then add a second instance of the same flow and use a configurable property to set a different value for the output queue.

If you want to create multiple instances of a message flow to improve performance, see "Optimizing message flow throughput" on page 2763.

Procedure

To deploy multiple instances of a message flow with different values for the configurable properties, complete the following steps:

1. In the Application Development view, create a copy of the message flow of which you want to deploy multiple instances; see "Copying a message flow or subflow by using copy" on page 580.
2. Add the original message flow and the copy of the message flow to your BAR file; see "Adding resources to a BAR file" on page 2470.
3. Open the BAR file.

   The resources in your BAR file are listed on the **Manage** tab.
4. On the **Manage** tab, you can now edit the configurable properties for both message flows.

   For more information, see "Editing configurable properties in a BAR file" on page 2477.

Results

Tip: The names that are assigned in the BAR file are also used on the command line when you run the **mqsilist** command on the integration server, or the **mqsichangeflowstats** command for a message flow.

What to do next

Deploy the BAR file by following the instructions in "Deploying integration solutions to a production environment" on page 2480. Both message flows are deployed and use the values for the configurable properties that you set in the BAR file.

Deploying integration solutions to a production environment

After you create and populate a BAR file with the development resources, you can deploy the integration solution to an integration server.

Before you begin

Complete the following steps:

- Review the deployment rules and guidelines; see "Deployment rules and guidelines" on page 2465.
- Package your integration solution in a BAR file; see "Packaging integration solutions" on page 2465.

Note:

The mode in which IBM App Connect Enterprise is running can affect the number of integration servers and message flows that you can deploy. For more information, see "Restrictions that apply in each operation mode" on page 5.

About this task

Choose one of the following methods to deploy a BAR file:

- "Deploying a BAR file by using the web user interface" on page 2480
- "Deploying a BAR file by using the IBM App Connect Enterprise Toolkit" on page 2481
- "Precompiling and deploying a BAR file by using the mqsibar command" on page 2481
- "Deploying a BAR file by using the mqsideploy command" on page 2482
- "Deploying a BAR file by using the IBM Integration API" on page 2483
- "Deploying a BAR file to IBM App Connect on IBM Cloud" on page 2484
- "Deploying a BAR file to IBM App Connect in containers" on page 2484

If you change a BAR file and want to propagate those changes to one or more integration servers, you must delete the existing BAR file from the integration server and deploy the new one.

Deploying a BAR file by using the web user interface

About this task

You can select a BAR file and deploy it to an integration server by using the web user interface.

You can also select an *overrides file*, which defines the BAR file properties that you want to change for a specific deployment. For more information, see command.

Procedure

1. Start the web user interface for your integration node or independent integration server, as described in "Accessing the web user interface" on page 89.

   For an integration node, the web user interface displays the node's integration servers. For an integration server, the web user interface displays the server's deployed message flows and other resources.
2. Select the **Deploy** action.

   For example:

   - In the web user interface for an integration node, select the required integration server (where you want to deploy the BAR file), and then click **Deploy**.
   - In the web user interface for an independent integration server, click **Deploy**.
3. Follow the instructions that are displayed.

Results

The contents of the BAR file are deployed to the integration server, and the deployed resources are shown.

Deploying a BAR file by using the IBM App Connect Enterprise Toolkit

About this task

To deploy a BAR file by using the IBM App Connect Enterprise Toolkit, complete the following steps. You can deploy to only one integration server at a time.

Procedure

1. Optional: Typically, an incremental BAR file deployment is done. To do a complete BAR file deployment, right-click the target integration server in the Integration Explorer view and click **Delete** > **All Flows and Resources**.

   Wait for the operation to complete before you continue.

   To refresh only some of the resources with the contents of the BAR file, do not click **Delete** > **All Flows and Resources**.
2. Deploy a BAR file to an integration server by using one of the following methods.

   - Drag the BAR file onto your target integration server in the Integration Explorer view.
   - Right-click the BAR file, then click **Deploy** and select the target integration server.
   - Right-click the target integration server, click **Deploy**, select **BAR from workspace** or **BAR from file system**, select the BAR file that you want to deploy, and click **OK**.
3. If the BAR file has changed since you last edited it, you are asked whether you want to save the file before deployment. If you click **Cancel**, the BAR file is not saved and deployment does not take place.

Results

The contents of the BAR file (for example, message flows and message sets) are deployed to the integration server. In the Integration Explorer view, the deployed resources are added to the appropriate integration server.

Precompiling and deploying a BAR file by using the `mqsibar` command

About this task

You can use the **`mqsibar`** command to deploy a BAR file to an integration server, by unpacking the contents directly into the integration server's `run` directory. You can also choose to precompile the

resources that the BAR file contains (such as XML schema, DFDL schema, and graphical data maps), so that they are ready to run when the integration server starts.

You can use the **mqsibar** command in a script, as part of an automated process that might include commands to test, tune, deploy, and run the application, as an alternative to deploying through the toolkit or web user interface. This method can be particularly useful if you are running IBM App Connect Enterprise in a cloud container.

Deploy a BAR file by using the **mqsibar** command, by completing the following steps.

Procedure

1. Open a command window that is configured for your environment.

   For information about configuring your command environment in preparation for running commands, see "Setting up a command environment" on page 64.
2. Run the **mqsibar** command, as shown in the following example:

   On Windows and Linux:

   Unpack the BAR file and compile resources into the integration server work directory, ready to be run when the integration server starts:

   ```
   mqsibar -a myAppAndShLibs.bar -c -w /sis01Wrk
   ```

   where:

   - **-a** specifies the name of the input BAR file to be processed.

     The BAR file must contain all referenced resources. If an application in the BAR file references a library that is contained by a different BAR file, an error occurs.
   - **-c** specifies that the contents of the BAR file will be compiled.

     Resources that are included in the BAR file, such as DFDL schema, XML schema, and graphical data maps, are compiled as a result of specifying this parameter.
   - **-w** specifies the integration server work directory.

     The integration server work directory is created by the **mqsicreateworkdir** command, which you run when you are configuring the integration server. When the work directory is created, a `run` subdirectory is also created, and it is this directory that the contents of the BAR file are unpacked into when the **mqsibar** command is run. For more information about configuring and starting an integration server, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172 and "Starting an integration server" on page 250.

   For more information about using the **mqsibar** command, see command.
3. Start (or restart) the integration server, for the changes to take effect.

Results

The command reports when responses are received from the integration server. If the command completes successfully, it returns 0 (zero).

Deploying a BAR file by using the `mqsideploy` command

About this task

To deploy a BAR file by using the **mqsideploy** command, complete the following steps.

Procedure

1. Open a command window that is configured for your environment.

2. Enter the appropriate command for your operating system and configuration, by using the following examples as a guide.

```
mqsideploy -i ipAddress -p port -a barfile
```

```
mqsideploy --admin-host ipAddress --admin-port port --bar-file barfile
```

The command completes an incremental deployment. To do a complete BAR file deployment, add the **-m** parameter.

The **-i** (IP address) and **-p** (port) parameters represent the connection details for the independent integration server or the integration node.

If deploying to an integration node, you must also specify the **-e** (integration server name) parameter. If you want to run this command against an integration node on the same computer, you can specify the integration node name instead of -i and -p, as in the following example:

```
mqsideploy integrationNodeName -e integrationServerName
-a barfile
```

If you have a `.broker` file that contains the connection details for an independent integration server or integration node, you can specify this file instead of -i and -p by using the **-n** parameter, as in the following example:

```
mqsideploy -n integrationNodeFileName -e integrationServerName
-a barfile
```

where *integrationNodeFileName* is the path and file name of the `.broker` file.

For alternative long names for parameters, and for more information about the command, see command.

Results

The command reports when responses are received from the integration node or independent integration server. If the command completes successfully, it returns 0 (zero).

Deploying a BAR file by using the IBM Integration API

About this task

To deploy a BAR file by using the IBM Integration API, use the `deploy` method of the `ExecutionGroupProxy` class.

The following code shows how an application can do an incremental deployment:

```
import
com.ibm.broker.config.proxy.*;
public class DeployBAR {

  public static void main(String[] args) {
    BrokerConnectionParameters bcp =
       new IntegrationNodeConnectionParameters("localhost",
4414);
    try {
      BrokerProxy b = BrokerProxy.getInstance(bcp);
      ExecutionGroupProxy eg = b.getExecutionGroupByName("default");
      DeployResult dr = eg.deploy("MyBAR.bar", true, 30000);
      System.out.println("Result = "+dr.getCompletionCode());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

By default, the `deploy` method completes an incremental deployment. To do a complete deployment, use a variant of the method that includes a `false` value for the Boolean `isIncremental` parameter. For example, `eg.deploy("MyBAR.bar", false, 30000)`.

Deploying a BAR file to IBM App Connect on IBM Cloud

With IBM App Connect on IBM Cloud (IBM App Connect on IBM Cloud - with the Lite plan or an Enterprise plan), you can also deploy and manage integration solutions that were developed in the App Connect Toolkit.

Before you begin

Before you can deploy an integration solution to IBM App Connect on IBM Cloud, you must sign up for an account with enterprise capabilities (the Lite plan or an Enterprise plan); see IBM App Connect in IBM Cloud.

About this task

App Connect on IBM Cloud provides a fully managed environment that you can use to deploy integration solutions to the cloud without the need to acquire and maintain an IT infrastructure.

You deploy a solution to App Connect on IBM Cloud by uploading the BAR file to the cloud. When you upload your BAR file to the cloud, an integration server is created in the cloud, and the contents of the BAR file are unpackaged and run on that integration server. Each integration server in App Connect on IBM Cloud contains the contents of a single BAR file. You can see your integration servers on the App Connect on IBM Cloud dashboard, where you can start and stop them. You can also configure authentication and secure connectivity between your resources in the cloud and on premises.

For detailed instructions about how to upload BAR files to the cloud, see Running enterprise integration solutions in App Connect on IBM Cloud.

What to do next

You can check the results of your deployment; see “Checking the results of deployment” on page 2485.

Deploying a BAR file to IBM App Connect in containers

Using IBM Cloud Pak for Integration or the App Connect Enterprise, you can deploy an IBM App Connect Enterprise Toolkit integration to run in an integration server in the container.

Before you begin

Before you can deploy an integration solution to IBM App Connect in containers, you must have installed IBM App Connect by using the IBM App Connect Operator, and created an instance of the App Connect Dashboard into which you want to deploy the BAR file.

- Ensure that you have access to the App Connect Dashboard.

About this task

You create the integration server by uploading your IBM App Connect Enterprise Toolkit integration in the form of a broker archive (BAR) file by using the App Connect Dashboard.

After you have uploaded your BAR file, you can create configurations that can be applied to the integration server when you deploy it, and can configure the integration server details.

The integration server is displayed as a tile on the **Servers** page of the dashboard, with an initial status of `Unavailable`, which then changes to `Started` when the deployment completes.

For detailed instructions about how to deploy BAR files to IBM App Connect in containers, see the following documentation:

- Deploying Toolkit integrations to IBM App Connect in containers (Continuous delivery)

- Deploying Toolkit integrations to IBM App Connect in containers (Extended Update Support)

What to do next

You can click the integration server tile to view the deployed integration. From the **Servers** page, you can also view the integrations for all listed integration servers by clicking **Integrations** to open the **Integrations** page.

Checking the results of deployment

After you deploy a BAR file, you can check that the operation completed successfully.

About this task

You can check the results of a deployment by using any of the following methods:

- "Checking deployment by using the IBM App Connect Enterprise Toolkit" on page 2485
- "Checking deployment by using the mqsideploy command" on page 2485
- "Checking deployment by using the IBM Integration API" on page 2486
- "Checking deployment to IBM Integration Bus on Cloud" on page 2486
- "Checking deployment to IBM App Connect on IBM Cloud" on page 2487

Also, check the system log on the target system where the integration node was deployed to make sure that the integration node did not report any errors.

If version or custom keyword information was included in the resources, you can also check the version of the resources that are deployed to your integration server; see "Viewing version information and custom keyword values in your deployed solutions" on page 319.

Checking deployment by using the IBM App Connect Enterprise Toolkit

About this task

Follow these steps to check a deployment by using the IBM App Connect Enterprise Toolkit:

Procedure

1. In the Integration Development perspective, open the Deployment Log view:
   a) Click **Window** > **Show View** > **Other**.
   b) In the **Show View** window expand **Integration Development**, and click **Deployment Log**.
2. View the messages that relate to each deployment in the Deployment Log view.

Results

The Deployment Log is updated only after a deployment is processed completely by the integration node. If a deployment fails, the reasons for the failure are shown here.

Checking deployment by using the `mqsideploy` command

About this task

If you use the **`mqsideploy`** command to deploy, the command displays the results of the deployment. For example, if you type the following command for the default integration server of the integration node:

```
mqsideploy INODE -e default -a my.bar
```

the following response indicates that the deployment was successful:

```
BIP1039I: Deploying BAR file 'my.bar' to integration node 'INODE' (integration
server 'default') ...
BIP1092I: The integration node successfully processed the deployment request.
```

The command also returns a numeric completion code value to indicate the outcome. If the deployment completes successfully, the command returns 0. For details of other values that you might see returned, see command.

Checking deployment by using the IBM Integration API

About this task

Code your application to test the results of the deployment actions that it takes. For example:

```
import com.ibm.integration.admin.model.common.LogEntry;
import com.ibm.integration.admin.model.server.DeployResult;
import com.ibm.integration.admin.proxy.IntegrationAdminException;
import com.ibm.integration.admin.proxy.IntegrationNodeProxy;
import com.ibm.integration.admin.proxy.IntegrationServerProxy;


public class DeployExample {

  public static void main(String[] args) {
    IntegrationNodeProxy node = new IntegrationNodeProxy("localhost",4414,"","",false);
    try {
      IntegrationServerProxy server = node.getIntegrationServerByName("server1");
      DeployResult result  = server.deploy("C:\\temp\\mybarfile.bar");
      int deployStatus = server.getLastHttpResponse().getStatusCode();
      if (deployStatus >= 400)
      {
        System.out.println("Http response error: " + deployStatus );
      }
      else
      { // deploy worked, check for any Log entries
        LogEntry[] logEntries  = result.getLogEntry();
        if (logEntries != null)
        {
          for (LogEntry logEntry: logEntries)
          {
            System.out.println("Deploy log entry: "+logEntry.getDetailedText());
          }
        }
      }
    } catch (IntegrationAdminException e) {
      System.out.println("Error deploying: "+e);
    }
  }
}
```

If the deployment message could not be sent to the integration node, an `IntegrationAdminException` exception is thrown at the time of the deployment. If the integration node receives the deployment message, log messages for the overall deployment are displayed, followed by completion codes specific to each integration node that is affected by the deployment.

Checking deployment to IBM Integration Bus on Cloud

About this task

When you upload BAR files into IBM Integration Bus on Cloud, a list of integrations is created. You can then start and stop those integrations, and view information about them.

To check the status of your integrations on the cloud, complete the following steps.

Procedure

1. Log in to IBM Integration Bus on Cloud.
   For more information, see Getting started.

2. Click **Integrations** to open the Integrations view.
3. Look at the status for the relevant integration.
   - If the status is `Started`, the BAR file has been deployed, and the integration is running.
   - If the status is `Stopped`, the BAR file has been deployed, but the integration has not yet been started.
   - If the status is `Preparing`, the BAR file is being deployed.

Checking deployment to IBM App Connect on IBM Cloud

About this task

When you upload a BAR file into IBM App Connect on IBM Cloud, an integration server is created to run the contents of that BAR file. The integration server is represented by a tile on the App Connect on IBM Cloud dashboard. You can start and stop the integration servers, and view information about them.

To check the status of your integrations on the cloud, complete the following steps.

Procedure

1. Log in to App Connect on IBM Cloud.
2. The App Connect on IBM Cloud dashboard should open automatically, but if it does not, click **Dashboard**.
3. Look at the status for the relevant integration server:
   - If the status is `Preparing`, the BAR file is being deployed.
   - If the status is `Stopped`, the BAR file has been deployed, but the integration server has not yet been started.
   - If the status is `Started`, the BAR file has been deployed, and the integration server is running.

What to do next

You can administer your integration node, integration server, and deployed resources; see Chapter 5, "Administering IBM App Connect Enterprise," on page 243.

Redeploying integration solutions to a production environment

If you change an integration solution, you can redeploy the updated solution to the same integration servers.

About this task

You can package an updated integration solution for redeployment by using one of the following methods:

- You can repackage the updated integration solution in the BAR file editor, and generate a new BAR file; see "Preparing packaged solutions for deployment" on page 2476.
- You can refresh the existing BAR file; see "Refreshing the contents of a BAR file" on page 2488.

You can redeploy the updated BAR file to one or more integration servers by using one of the deployment methods that are described in "Deploying integration solutions to a production environment" on page 2480. You do not have to stop the message flows that you deployed previously. All the resources in the integration server that are also in the redeployed BAR file are replaced.

You do not have to redeploy JAR files unless you updated them. If one or more JAR files in your BAR file are present on the computer where the integration node is running, you can remove them from your BAR file before you deploy the BAR file again.

If your updates to the BAR file include the deletion of resources, a redeployment does not result in their deletion from the integration node. For example, assume that your BAR file contains applications A1, A2,

and A3. Update the file by removing A2 and adding application A4. If you redeploy the BAR file, all four applications are available in the integration server when the redeployment completes. A1 and A3 are replaced by the contents of the redeployed BAR file.

To clear previously deployed resources from the integration server before you redeploy (for example, if you are deleting resources), use one of the methods that are described in "Deleting deployed resources" on page 319.

- To use the IBM App Connect Enterprise Toolkit, follow the instructions for a new deployment, making sure that you select **Delete** > **All Flows and Resources** before you deploy resources.
- To use the **mqsideploy** command, follow the instructions for a new deployment, making sure that you add the **-m** parameter to perform a complete BAR file deployment.
- To use the IBM Integration API, follow the instructions for a new deployment.

If your message flows are not transactional, stop the message flows before you redeploy resources to ensure that all the applications complete cleanly and are in a known and consistent state. You can stop individual message flows, integration servers, or integration nodes.

If your message flows are transactional, the processing logic that handles commitment or rollback ensures that resource integrity and consistency are maintained.

What to do next

Check the results of the redeployment by following the instructions in "Checking the results of deployment" on page 2485.

Refreshing the contents of a BAR file

Refresh the contents of a BAR file by rebuilding it in the BAR file editor.

Before you begin

You must have created a BAR file and added files to it. For more information, see "Creating a BAR file" on page 2469 and "Adding resources to a BAR file" on page 2470.

About this task

If you change resources that you added to your BAR file, the BAR file is inconsistent, and is shown with an "out-of-sync" icon in the Application Development view. The BAR file is also inconsistent if any changes are made to the container (integration project, application, or library) to which the resources belong. For example, the BAR file is inconsistent if a new message flow is added to an application where existing resources from the application are in the BAR file.

If you want to rename a deployed resource, see "Renaming resources that are deployed to integration servers" on page 2489.

To refresh the contents of a BAR file before you deploy it, complete the following steps.

Procedure

1. Open the BAR file.
   The contents of the BAR file are shown on the **Manage** tab of the BAR File editor.

2. To refresh all the resources in the BAR file on the **Manage** tab, click **Rebuild existing BAR entries**  .

   A dialog box opens, showing progress. When the operation is complete, click **Details** to see information about what was refreshed.

   Alternatively, you can refresh the archive contents by right-clicking a BAR file in the Application Development view and selecting **Build and Save BAR**. The BAR file is rebuilt in the background.

   You can view the user and service logs on the **User Log** and **Service Log** tabs of the BAR File editor.

   Clear the contents of logs by clicking **Remove** ().
3. Optional: To view details about the build of an individual deployable resource on the **Manage** tab, right-click the resource and click **Details**.

   The Properties view opens and the **Details** tab is displayed. The **Details** tab shows the following details about the deployable resource:

   - The workspace resource, with references to the linked workspace resources (for example, `.msgflow` `.mset`, `.xml`, and `.xslt` files).
   - The status of the last compilation, which shows the user log entry for the last compilation.

What to do next

Deploy the BAR file by following the instructions in "Deploying integration solutions to a production environment" on page 2480 or "Redeploying integration solutions to a production environment" on page 2487.

Renaming resources that are deployed to integration servers

You cannot rename a resource while it is still deployed to an integration server. You must change the name of the resource that is in the BAR, then redeploy the BAR file.

Before you begin

You must have deployed resources to an integration server.

About this task

To rename a deployed resource, complete the following steps.

Procedure

1. Remove the deployed resource from the integration server.

   For more information, see "Deleting deployed resources" on page 319.
2. Right-click the development resource.
3. Optional: If you rename a resource that is referenced by another resource, you might need to update the references to the renamed resource.
   For example, suppose you rename a library that is referenced by an application. Renaming the library breaks the reference between the application and library. To fix the reference, right-click the application, click **Manage library references**, and select the renamed library.
4. Refresh the contents of the BAR file by clicking **Rebuild and save BAR**  on the **Manage** tab of the BAR File editor.
5. Redeploy the BAR file.

Chapter 8. Security

Security is an important consideration for both developers of IBM App Connect Enterprise applications, and for system administrators configuring IBM App Connect Enterprise authorities.

When you are designing an IBM App Connect Enterprise application, it is important to consider the security measures that are needed to protect the information in the system.

The “Security overview” on page 2491 introduces the concepts that you need to understand before you set up security for IBM App Connect Enterprise.

The following topics explain the different aspects of security that you need to consider when you are designing your applications:

Security overview

When you are designing an IBM App Connect Enterprise application it is important to consider the security measures that are needed to protect the information in the system.

An important aspect of securing an enterprise system is the ability to protect the information that is passed from third parties. This capability includes restricting access to IBM MQ and JMS queues. For information about the steps involved, refer to the documentation supplied by your transport provider. If you are using HTTPS, you need to set specific properties in the HTTP nodes. For information about this option, see node, node, and node.

In addition to securing the transport, you can secure individual messages based on their identity. For more information about securing messages in a message flow, see “Message flow security” on page 2550.

Some security configuration is required to enable IBM App Connect Enterprise to work correctly and to protect the information in the system. For example, you can secure the messaging transport with SSL connections, restrict access to queues, apply WS-Security to web services, and secure access to message flows.

The following topics introduce the concepts that you need to understand before you set up security for IBM App Connect Enterprise:

Authorization for configuration tasks

Authorization is the process of granting or denying access to a system resource.

For IBM App Connect Enterprise, authorization is concerned with controlling who has permission to access the integration server and its resources, and ensuring that users who attempt to work with them have the necessary permissions to do so.

Examples of tasks that require authorization are:

- Configuring an integration server; for more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
- Accessing queues; for example, putting a message to the input queue of a message flow.
- Taking actions within the IBM App Connect Enterprise Toolkit; for example, deploying a message flow to an integration server.

Security exits

Use security exit programs to verify that the partner at the other end of a connection is genuine.

When you connect from the IBM App Connect Enterprise Toolkit to an integration server on another computer, a security exit is not started by default to monitor the connection. If you want to protect access to the integration server from client programs, you can use the IBM MQ security exit facility.

You can enable a security exit at each end of the connection between your client session and the integration server :

- Set up a security exit on the channel at the integration server end. This security exit has no special requirements; you can provide a standard security exit.
- Set up a security exit in the IBM App Connect Enterprise Toolkit. Identify the security exit properties when you connect to the integration server . The security exit is a standard IBM MQ security exit, written in Java.

For an overview of security exits and details of their implementation, see "Channel security exit programs" in the *Intercommunication* section of the IBM MQ product documentation online.

Public key cryptography

All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on private key encryption. Public key encryption is the only challenge to private key encryption that has appeared in the last 30 years.

Private key encryption

Private key encryption systems use a single key that is shared between the sender and the receiver. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both the sender and receiver must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread general use:

- Private key encryption requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- Keys must be shared between pairs of communicators, therefore the keys must be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. You cannot send a usable encrypted message to someone spontaneously. You and the other participant must make arrangements to communicate by sharing keys.

Private key encryption is also called symmetric encryption because the same key is used to encrypt and decrypt the message.

Public key encryption

Public key encryption uses a pair of mathematically-related keys. A message that is encrypted with the first key must be decrypted with the second key, and a message that is encrypted with the second key must be decrypted with the first key.

Each participant in a public key system has a pair of keys. One key is nominated as the private key and is kept secret. The other key is distributed to anyone who wants it; this key is the public key.

Anyone can encrypt a message by using your public key, but only you can read it. When you receive the message, you decrypt it by using your private key.

Similarly, you can encrypt a message for anyone else by using their public key, and they decrypt it by using their private key. You can then send the message safely over an unsecured connection.

This kind of encryption has characteristics that make it very suitable for general use:

- Public key encryption requires only two keys per participant.
- The need for secrecy is more easily met: only the private key needs to be kept secret, and because it does not need to be shared, it is less vulnerable to theft in transmission than the shared key in a symmetric key system.
- Public keys can be published, which eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public key encryption is also called asymmetric encryption, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other.

With symmetric key encryption, beware of stolen or intercepted keys. In public key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying the identity of the owner of the public key. Nothing prevents a user from creating a key pair and publishing the public key under a false name. The listed owner of the public key cannot read messages that are encrypted with that key because the owner does not have the corresponding private key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages that are intended for someone else. To counteract the potential for forged keys, public key systems provide mechanisms for validating public keys and other information with digital certificates and digital signatures.

Public Key Infrastructure (PKI)

PKI is an infrastructure that uses public key technology to allow applications to interact securely. PKI uses public key encryption to provide privacy. In practice, only a small amount of data is encrypted in this way. Typically, a *session key* is used with a symmetric algorithm to transmit the bulk of the data efficiently.

In business transactions, trust is even more important than privacy. PKI uses the private key to allow an application to sign a document. For the recipient to authenticate the sender, it needs a reliable way to obtain the public key for the sender. This public key is provided in the form of a digital certificate, which is mediated by a trusted third party certificate authority (CA).

Digital certificates

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

A digital certificate is equivalent to an electronic ID card. The certificate serves two purposes:

- Establishes the identity of the owner of the certificate
- Distributes the owner's public key

Certificates are issued by trusted third parties, called certificate authorities (CAs). These authorities can be commercial ventures or local entities, depending on the requirements of your application. The CA is trusted to adequately authenticate users before issuing certificates. A CA issues certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized

as intact and authentic. Participants in an application need to validate certificates only; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The designation, "trusted third parties", indicates that the system relies on the trustworthiness of the CAs.

The certificates and private keys are stored in files called *keystores* and *truststores*.

- A keystore holds the private keys and public key certificates for an application.
- A truststore contains the CA certificates required to authenticate certificates that are presented by another application.

Contents of a digital certificate

A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:

- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but also the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate is issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. The message-digest function creates a signature based upon all the previously listed fields.

The idea of a certificate is that a CA takes the public key of the owner, signs the public key with its own private key, and returns the information to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate that contains the CA signature with the public key of the owner. By using the CA public key and the CA signature on the extracted certificate, the receiver can validate the CA signature. If valid, the public key that is used to extract the certificate is considered good. The owner signature is validated, and if the validation succeeds, the owner is successfully authenticated to the receiver.

The additional information in a certificate helps an application to determine whether to honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

An application that needs to authenticate itself must provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a signer certificate. In cases where chains of trust are established, several signer certificates can be involved.

Requesting certificates

To get a certificate, send a certificate request to the CA. The certificate request includes:

- The distinguished name of the owner or the user for whom the certificate is requested
- The public key of the owner
- The digital signature of the owner

The message digest function creates a signature based on all the previously listed fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is authenticated successfully, the CA issues a certificate for that owner.

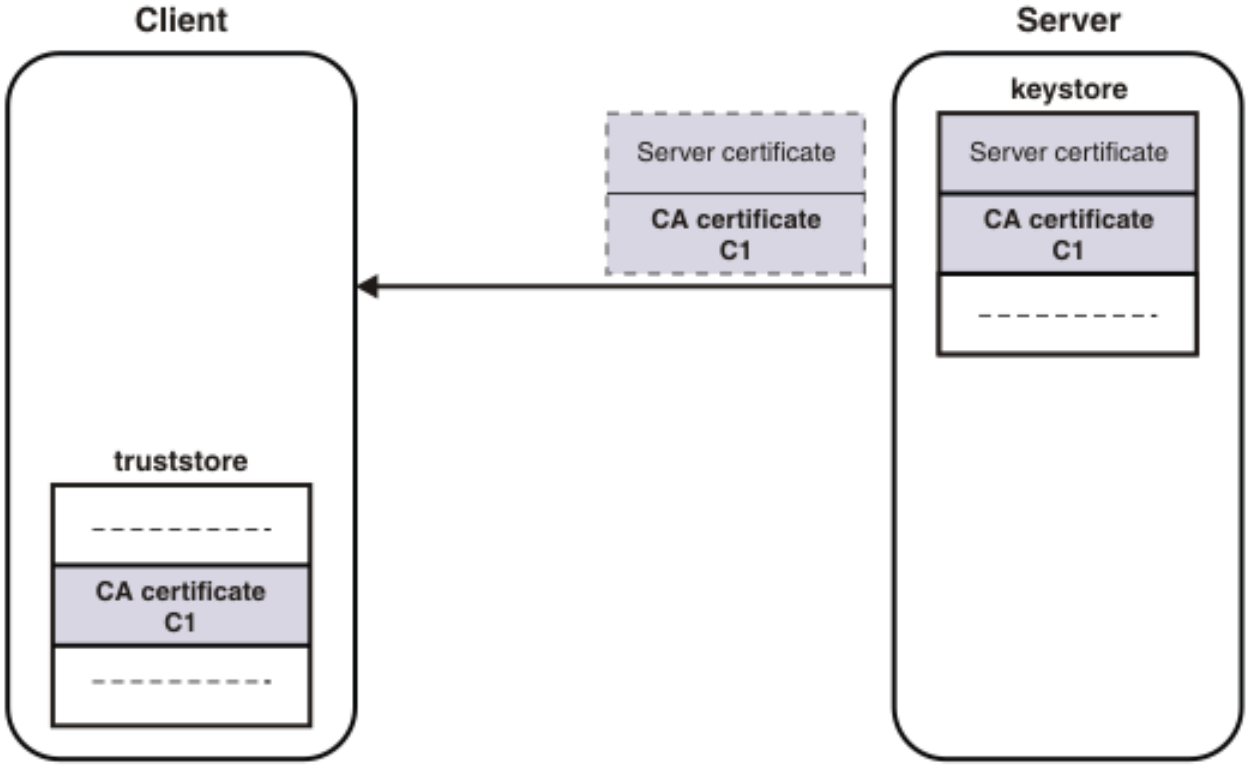
Using certificates: Chain of trust and self-signed certificate

To verify the digital signature on a certificate, you must have the public key of the issuing CA. Public keys are distributed in certificates, therefore you must have a certificate for the issuing CA that is signed by the issuer. One CA can certify other CAs, so a chain of CAs can issue certificates for other CAs, all of whose public keys you need. Eventually, you reach a root CA that issues itself a self-signed certificate. To validate a user certificate, you need certificates for all of the intervening participants back to the root CA. You then have the public keys that you need to validate each certificate, including the user certificate.

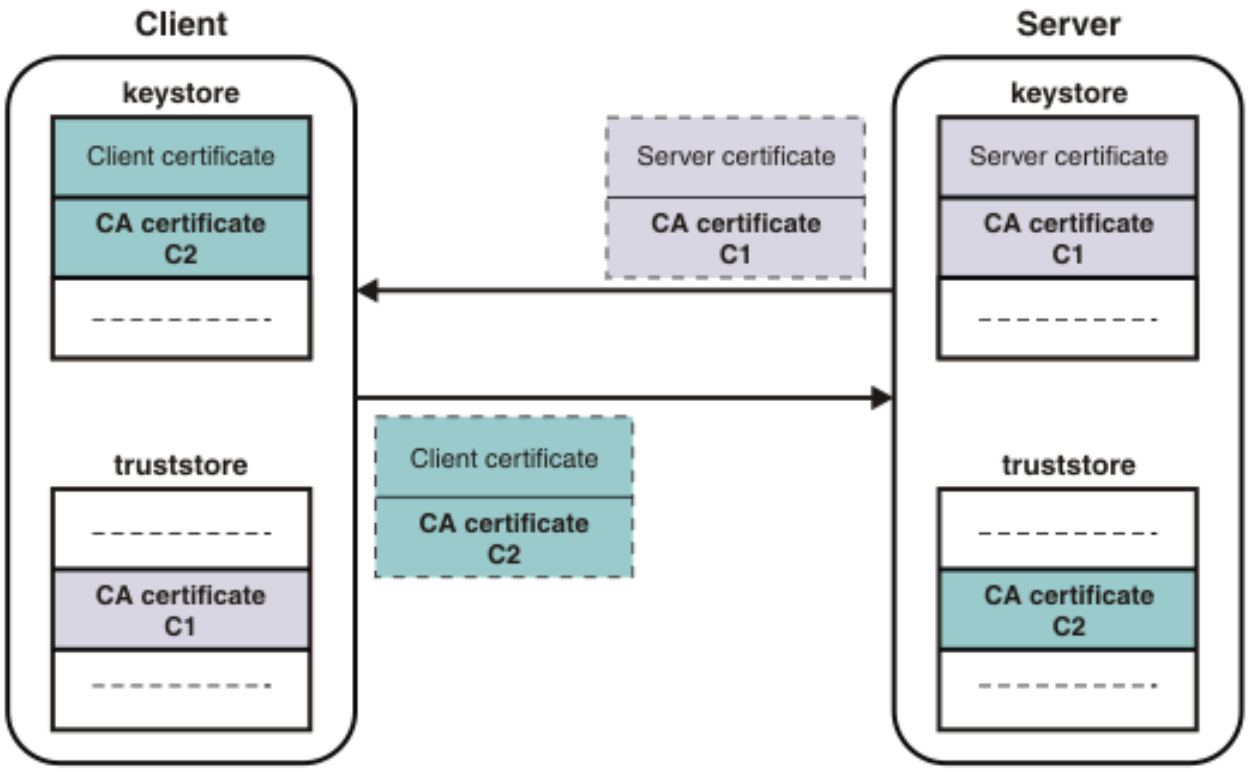
A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is valid, the public key it contains is used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you can probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is a trusted certificate authority. To ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means; for example, hand-delivered on floppy disks, downloaded from secure sites, and so on.

Applications that use certificates store these certificates in a keystore file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server when mutual authentication is enabled.

To allow a client to authenticate a server, a server keystore file contains the private key and the certificate of the server and the certificates of its CA. A client truststore file must contain the signer certificates of the CAs of each server, which the client must authenticate. The following diagram illustrates how a client authenticates a server.



If mutual authentication is needed, the client keystore file must contain the client private key and certificate. The server truststore file requires a copy of the certificate of the client CA. The following diagram illustrates mutual authentication.



Digital signatures

A digital signature is a number that is attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes the following information:

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message signed by the user who claims to have signed it?

A digital signature is created in two steps. The first step distills the document into a large number. This number is the digest code or fingerprint. The digest code is then encrypted, which results in the digital signature. The digital signature is appended to the document from which the digest code is generated.

Several options are available for generating the digest code. This process is not encryption, but a sophisticated checksum. The message cannot regenerate from the resulting digest code. The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

To stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code, you need a way to verify the digest code as well. To verify the digest code, reverse the use of the public and private keys. For private communication, it makes no sense to encrypt messages with your private key; these keys can be decrypted by anyone with your public key. However, this technique can be useful for proving that a message came from you. No one can create it because no-one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, the decryption process verifies that the holder of the corresponding private key did encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code is computed for a document, the digest code is encrypted with the sender's private key. The result is the digital signature, which is attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:

1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.
3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Administration security

Administration security controls the rights of users to complete administrative tasks for integration nodes, integration servers, and other IBM App Connect Enterprise resources.

The following topics introduce the concepts that you need to understand before you can set up administration security for IBM App Connect Enterprise, and explain the steps involved in securing integration nodes and their resources:

- “Administration security overview” on page 2497
- “Setting up administration security” on page 2504

You can also control access to independent integration servers and integration nodes by using either HTTP basic authentication or by using SSL or TLS. The following topics explain how to configure basic authentication and SSL or TLS:

- “Configuring HTTP basic authentication for an integration node or server” on page 2548
- “Configuring SSL or TLS for an integration node or server” on page 2548

Administration security overview

Administration security controls users' permissions to access integration nodes, integration servers, and their resources, and to complete administrative tasks.

Administration security is an optional feature of IBM App Connect Enterprise; it is not enabled by default. You can enable authentication and select the required authorization mode, either by using the **mqsichangeauthmode** command or by setting properties in the `server.conf.yaml` or `node.conf.yaml` configuration files. For more information, see “Enabling administration security” on page 2504.

You can control users' access to integration nodes, integration servers, and their resources by associating users with roles. A role is a set of security permissions that control access to runtime components and resources, and each user account is associated with a particular role. The permissions are checked to determine a user's authorization to perform tasks through the web user interface, REST API, IBM App Connect Enterprise Toolkit, and IBM App Connect Enterprise commands. For more information about roles, see “Role-based security” on page 2503, and for information about how to create and assign roles to web users, see “Managing web user accounts” on page 2510.

The following aspects of administration security are supported by IBM App Connect Enterprise:

- Authentication
- Authorization

Authentication

Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. IBM App Connect Enterprise provides authentication support for the following administration interfaces:

- IBM App Connect Enterprise web user interface

- IBM App Connect Enterprise RESTful application programming interface (API)
- IBM App Connect Enterprise Toolkit
- IBM App Connect Enterprise commands. You can use some commands to provide security credentials when you are connecting to a remote host, by providing an ID and password as a URI, on the **-i** *URI* parameter.

For these administration interfaces, authentication is done by App Connect Enterprise. To use App Connect Enterprise to authenticate a user, you create a web user account with a local password. The user ID and password are then checked against the credentials that are held in the system.

Alternatively, you can use an external LDAP server to authenticate a user. To use an LDAP server, you must configure your integration node to use an LDAP server for authentication. Credentials are checked against the username and password set in the LDAP server. When LDAP is enabled, local passwords are ignored. Unless you intend to use file-based authorization, you do not need to create a web user account with a local password.

For more information about the authentication support that is provided by IBM App Connect Enterprise, see "Authenticating users for administration" on page 2507.

Authorization

Authorization is the process of controlling users' access to resources, by verifying that they have the necessary permissions to complete the requested actions against the specified resources.

When administration security is enabled, you can control users' access to integration nodes, integration servers, and resources. Control the access by setting permissions that allow user IDs associated with specified roles to complete actions on specified resources. App Connect Enterprise checks the authorizations when it receives a request to view or change its properties or resources. If the user ID associated with the request is not authorized, the request is denied. Permissions are checked for all actions that are attempted by users of the following interfaces:

- IBM App Connect Enterprise
- IBM App Connect Enterprise Toolkit sessions
- IBM App Connect Enterprise RESTful application programming interface (API)
- Java programs that use the IBM Integration API to complete operations on the integration node.
- All the following commands:
  - **mqsichangeresourcestats**
  - **mqsicreateexecutiongroup**
  - **mqsideleteexecutiongroup**
  - **mqsideploy**
  - **mqsilist**
  - **mqsimode**
  - **mqsireloadsecurity**
  - **mqsireportresourcestats**
  - **mqsistartmsgflow**
  - **mqsistopmsgflow**
  - **mqsiwebuseradmin**

  For more information about authorization that is required for these commands, see Commands and authorizations for administration security.

  You can run all commands that are not stated here only on the computer on which the integration node or server is running. When you run any unlisted commands, the user ID that is used to run the commands must be a member of the security group mqbrkrs. Alternatively, it must be the same user ID that is running the integration node or server.

Users of the web user interface and the IBM App Connect Enterprise Toolkit who do not have read, write, and execute permissions for the integration node or integration servers, have only restricted access to those resources.

For **integration nodes** and their managed integration servers, three modes of authorization are supported: file-based authorization (*file* mode), queue-based authorization (*mq* mode), and LDAP authorization mode. You can enable administration security for an integration node and specify file-based authorization or queue-based authorization mode, by using either of the following methods:

- Use the **mqsichangeauthmode** command, as described in "Configuring authorization by using the mqsichangeauthmode command" on page 2528.
- Set the properties in the integration node's `node.conf.yaml` configuration file, as described in "Configuring authorization for an integration node by modifying the node.conf.yaml file" on page 2523.

You can specify LDAP authorization by setting properties in the integration node's configuration file, as described in "Configuring authorization by using LDAP groups" on page 2537.

For independent **integration servers**, file-based authorization (*file* mode), and LDAP authorization are supported. You can enable file-based authorization for an independent integration server (which is not managed by an integration node), by using one of the following methods:

- Use the **mqsichangeauthmode** command as described in "Configuring authorization by using the mqsichangeauthmode command" on page 2528.
- Set the properties in the integration server's `server.conf.yaml` configuration file, as described in "Configuring authorization for an integration server by modifying the server.conf.yaml file" on page 2526.

You can enable LDAP authorization for an independent integration server (which is not managed by an integration node), by setting properties in the integration server's `server.conf.yaml` configuration file, as described in "Configuring authorization by using LDAP groups" on page 2537.

File-based authorization (*file* mode)

File-based authorization can be configured for integration nodes (and their managed integration servers) and for independent integration servers.

If an integration node or server is configured to use file-based authorization, you can grant permissions to a role by using the command, or by setting permissions in the `node.conf.yaml` or `server.conf.yaml` configuration file. For more information, see and "Setting file-based permissions" on page 2530.

If no permissions are found for the role name, a check is conducted to see if the role name matches a system user ID. If a matching system user ID exists, and if it is a member of the mqbrkrs group, full permissions are given.

Queue-based authorization (*mq* mode)

Queue-based authorization can be configured for integration nodes (and their managed integration servers).

If the queue-based mode of authorization is set for an integration node, you specify permissions on authorization queues, which are defined on the queue manager that is specified on the integration node:

- SYSTEM.BROKER.AUTH. This queue represents the integration node and its properties. Only one queue exists of this name for each integration node. This queue is defined as a local queue.
- One SYSTEM.BROKER.AUTH.*EG* for each integration server that you define on the integration node, where *EG* is the name of the integration server. These queues are defined as alias queues.

Read, write, and execute authorities are granted automatically to the user group mqbrkrs on the SYSTEM.BROKER.AUTH queue.

When you create an integration server on an integration node for which security is enabled, the integration server authorization queue SYSTEM.BROKER.AUTH.*EG* is created, where *EG* is the name of the integration server. Read, write, and execute authorities are automatically granted to the user group mqbrkrs on this queue.

If the integration node is configured to use queue-based authorization, you must create a system user ID on the operating system on which your integration node is running. You then assign permissions to the system user ID, and this set of permissions represents a role with a name that corresponds to the name of the system user ID. For example, the set of permissions that you define for a system user who is called `ibmuser` for a role that is called `ibmuser`. For more information about setting permissions for queue-based authorization, see "Setting queue-based permissions" on page 2533.

LDAP authorization mode

LDAP authorization can be configured for integration nodes (and their managed integration servers) and for independent integration servers. If an integration node or server is configured to use LDAP authorization, you can grant permissions to a role by setting permissions in the `node.conf.yaml` or `server.conf.yaml` configuration file. For more information, see "Configuring authorization by using LDAP groups" on page 2537.

For more information about the authorization support provided by IBM App Connect Enterprise, see "Authorizing users for administration" on page 2516 and "Role-based security" on page 2503.

For more information about security permissions, see the following topics:

- "Permissions for acting on integration nodes, integration servers, and resources" on page 2518
- "Permissions for connecting to a queue manager" on page 2521
- "Setting file-based permissions" on page 2530
- "Setting queue-based permissions" on page 2533
- "Authorization queues for queue-based administration security" on page 2501
- "Role-based security" on page 2503

IBM App Connect Enterprise permissions and equivalent IBM MQ permissions and LDAP permissions

If you have enabled integration node administration security, you can give different permissions to user IDs to allow them to complete various actions against an integration node or its resources.

When a user requests an action against an integration node or an integration server, the integration node accesses the stored authorization information to check whether the user ID has the required permissions for that action against the target resource.

If queue-based administration security is enabled on the integration node, permissions are specified on authorization queues. Permission to perform an integration node administration task is mapped to an IBM MQ authority associated with the relevant authorization queue, and is created and maintained by the integration administrator.

If file-based security is enabled on the integration node, permissions are specified using the **`mqsichangefileauth`** command, and can be created and maintained by user IDs that are members of the mqbrkrs group.

If LDAP security is enabled on the integration node, permissions are specified by setting properties in the `node.conf.yaml` configuration file.

The mapping from integration node permission to IBM MQ permission, file-based permission and LDAP permission is shown in the following table.

Integration node permission	**IBM MQ permission**	**MQ queue-based permission (using `setmqaut`)**	**File-based permission (using `mqsichangefileauth`) or LDAP permission (using .conf.yaml configuration file)**
Read	Inquire	+INQ	read+
Write	Put	+PUT	write+

Integration node permission	IBM MQ permission	MQ queue-based permission (using `setmqaut`)	File-based permission (using `mqsichangefileauth`) or LDAP permission (using .conf.yaml configuration file)
Execute	Set	+SET	set+

For information about the authorizations that are required for specific tasks, see Tasks and authorizations for administration security.

IBM MQ specific and generic profiles

IBM MQ supports both specific and generic profiles to manage IBM MQ permissions. When you enable queue-based administration security, you can create specific profiles to define IBM MQ permissions on SYSTEM.BROKER.AUTH and on one or more SYSTEM.BROKER.AUTH.*EG* queues (where *EG* is the name of a specific integration server).

You might want to grant a user, or group of users, authority to a number of integration servers, or perhaps all integration servers. You can use an IBM MQ generic profile to grant authority in this way. A generic profile defines authority to an existing set of integration servers, and all additional groups, that match the profile. A generic profile is one that uses special characters (wildcard characters) in the profile name, such as asterisks (*).

For example, if you want to create a generic profile to authorize access to all integration servers defined on the integration node, you can specify SYSTEM.BROKER.AUTH.**. If you want a profile for a set of integration servers with names that all start with the same character string, you can specify SYSTEM.BROKER.AUTH.TEST**.

For more information about IBM MQ generic profile wildcard characters, see Wildcards used in generic profiles, and for information about IBM MQ generic profile priorities, see Profile priorities in the IBM MQ product documentation online.

Authorization queues for queue-based administration security

If the queue-based mode of administration security is enabled, the integration node examines specific queues to determine whether a user has the required permissions to complete a particular task against the integration node or its resources.

If a queue manager is specified on the integration node, the queue-based mode of administration security (*mq* mode) is specified by default. For more information about authorization modes, see “Configuring administration security to use file-based, queue-based, or LDAP authorization” on page 2523.

You set security permissions on the following authorization queues:

SYSTEM.BROKER.AUTH
: The queue SYSTEM.BROKER.AUTH is created when you use the **mqsichangeauthmode** command to enable queue-based administration security (`mq` mode) on the integration node. Read, write, and execute permissions are granted automatically to the user group mqbrkrs on this queue. The SYSTEM.BROKER.AUTH queue is created as a local queue, and is used to define which users are authorized to perform actions on the integration node and its properties.

  If the **mqsichangeauthmode** command fails to create the queue for any reason, you can create it manually; see “Creating the default system queues on an IBM MQ queue manager” on page 99.

SYSTEM.BROKER.AUTH.*integrationServerName*
: When you create an integration server on an integration node for which you have enabled queue-based administration security, the integration server authorization queue SYSTEM.BROKER.AUTH.*integrationServerName* is created (if it did not already exist), where *integrationServerName* is the name of the integration server. If the queue already exists, check that the permissions are appropriate. Read, write, and execute permissions are automatically granted to

the user group mqbrkrs on this queue. The dedicated integration server queues are created as aliases to the queue SYSTEM.BROKER.AUTH.

If the integration node fails to create the queue for any reason, you can create it manually; see "Creating the default system queues on an IBM MQ queue manager" on page 99.

SYSTEM.BROKER.DC.AUTH

When you use the **mqsicreatebroker** command to create an integration node with an associated queue manager, the SYSTEM.BROKER.DC.AUTH queue is created automatically. If you create the integration node without specifying a queue manager, and subsequently modify it to specify a queue manager and enable queue-based administration security (`mq` mode), you must also create the SYSTEM.BROKER.DC.AUTH queue; see "Creating the default system queues on an IBM MQ queue manager" on page 99.

SYSTEM.BROKER.DC.AUTH.*integrationServerName*

When you create an integration server on an integration node for which you have enabled queue-based administration security, the integration server authorization queue SYSTEM.BROKER.DC.AUTH.*integrationServerName* is created (if it did not already exist), where *integrationServerName* is the name of the integration server. If the queue already exists, check that the permissions are appropriate. Read, write, and execute permissions are automatically granted to the user group *mqbrkrs* on this queue. The dedicated integration server queues are created as aliases to the queue SYSTEM.BROKER.DC.AUTH.

If the integration node fails to create the queue for any reason, you can create it manually; see "Creating the default system queues on an IBM MQ queue manager" on page 99.

If you create an integration node without enabling administration security, you can change it later by using the **mqsichangeauthmode** command. If you have defined one or more integration servers on that integration node when you change its security setting, the required integration server authorization queues are defined.

A queue can be created only by a user ID that is a member of the IBM MQ security group mqm. Therefore, the user ID that is used to create or change an integration node, and the ID under which the integration node is running when an integration server is created, must be a member of that security group. If the user ID does not have the required permissions, a message is returned to the command , or written to the system log, with the error and the name of the queue. You must create the queue yourself, or ask your IBM MQ administrator to create it for you.

IBM MQ restricts the length of a queue name to 48 characters. Queue name characters must be in the En_US ASCII character set, and contain only uppercase and lowercase letters, digits, and the following special characters: period (.), forward slash (/), underscore (_), and percent (%). If the name of your integration server includes a character that is not valid, that character is replaced in the IBM MQ queue name by an underscore character. For example, if you create an integration server with the name `test@environment`, the authorization queue is created with the name SYSTEM.BROKER.AUTH.test_environment.

If you are running a secure environment, limit the names of your integration servers to 26 characters. This limit ensures that the authorization queue names generated, which include the prefix SYSTEM.BROKER.AUTH, do not exceed the IBM MQ limit of 48 characters.

If your integration server names do not all conform to the length and character requirements, integration servers with similar names might result in a shared authorization queue. If this situation occurs, a warning message is returned to the user that issued the command, or is written to the system log, when the second integration server is created to state that the queue is shared.

When you delete an integration server, its associated authorization queue is retained. The queue is deleted if you specify the appropriate parameter when you delete the integration node. The queue can be reused if you recreate the integration server, but you must check the permissions that you have defined on the queue to ensure that they are still valid.

If you rename an integration server, you must first create an authorization queue with the appropriate name. You must also recreate the IBM MQ permissions associated with the original authorization queue

on this queue before you rename the integration server; the integration node does not perform this task on your behalf. The integration node rejects the rename request if the authorization queue does not exist, to ensure that security is not affected by the renaming. If you do not recreate these permissions, no user IDs are authorized to perform a task against the renamed integration server.

When you delete an integration node, you can specify that all its authorization queues are also deleted; they are not deleted by default.

For more information about setting permissions for queue-based authorization, see "Setting queue-based permissions" on page 2533.

Role-based security

You can control access to integration nodes, integration servers, and their resources through the web user interface, REST application programming interface (API), IBM App Connect Enterprise Toolkit, and App Connect Enterprise commands, by associating users with roles.

A *role* is defined by a set of security permissions that control users' access to an integration node or integration server, and its associated resources.

As an integration administrator, you can control the access that web users have to integration nodes, integration servers, and resources, by assigning each user to a role. You can authorize users with a particular role to complete specific actions; for example, you might allow users with one role to view integration server resources, while allowing users with another role to modify them.

You can grant the same permissions to multiple users by assigning them to the same role, but each user can be assigned to only one role.

You can configure integration nodes (and their associated integration servers) to use file-based authorization, queue-based authorization, or LDAP authorization. You can configure independent integration servers (which are not associated with an integration node) to use file-based authorization or LDAP authorization. For information about how to set the authorization mode, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

If an integration node or an independent integration server is configured to use file-based authorization (**file** mode), you can grant permissions to a role either by using the **-r** *role* and **-p** *permissions* parameters of the command, or by setting permissions in the `node.conf.yaml` or `server.conf.yaml` configuration files. For more information about file-based authorization, see "Setting file-based permissions" on page 2530.

In **file** mode, if a user is assigned to a role that has no permissions defined, when that user attempts an action, a check is made to see whether the role name matches a local operating system user name. If there is a match (for example, both the role name and operating system user name are `aceadmin`), a check is made to see whether that user name is a member of the mqbrkrs group. If it is a member, permission is granted for all actions on all objects.

If an integration node is configured to use queue-based authorization (**mq** mode), you must create a system user name on the operating system on which your integration node is running. You then assign permissions to the system user name, and this set of permissions represents a role with a name that corresponds to the name of the system user name. For example, the set of permissions that you define for a system user called `ibmuser` form a role called `ibmuser`. For information about setting permissions for queue-based authorization, see "Setting queue-based permissions" on page 2533.

If an integration node or an independent integration server is configured to use LDAP authorization (**ldap** mode), you can grant permissions to a role by setting permissions in the `node.conf.yaml` or `server.conf.yaml` configuration files. For more information about LDAP authorization, see "Configuring authorization by using LDAP groups" on page 2537.

You can create web user accounts and assign them to the appropriate roles by using the command. For more information, see "Managing web user accounts" on page 2510 and "Controlling access to data and resources in the web user interface" on page 2544.

Setting up administration security

Control the actions that users can perform on an integration node or integration server, and its resources.

About this task

You can enable administration security and specify the required authorization mode for an integration node or integration server by using the command or in the node.conf.yaml or server.conf.yaml configuration file. When you have activated administration security, grant users or groups the required permissions to complete their tasks.

You can grant permissions to users of the web user interface by associating the web user account with a predefined role. For more information, see “Role-based security” on page 2503 and “Managing web user accounts” on page 2510.

Procedure

To set up administration security for an integration node or integration server, complete the following steps:

1. Enable administration security and set the required authorization mode.

   For more information, see “Enabling administration security” on page 2504.
2. Control access to the administration interfaces by using the authentication capabilities that are provided with IBM App Connect Enterprise.

   For more information, see “Authenticating users for administration” on page 2507 and “Managing web user accounts” on page 2510.
3. Set the appropriate permissions to authorize users to complete specific tasks.

   For more information, see “Authorizing users for administration” on page 2516.

Enabling administration security

Enable administration security on an integration node or integration server, to control which users can complete specific tasks against that integration node or server and its resources.

About this task

If you do not enable administration security for an integration node, all users are able to complete all actions against the integration node and all its managed integration servers. If you do not enable administration security for an independent integration server (which is not managed by an integration node), all users are able to complete all actions against that integration server and its resources. If administration security is not enabled, web users can access the web user interface as the default user, with unrestricted access to data and resources.

You can enable administration security and specify the authorization mode for an integration node or integration server, either by setting the **`authorizationEnabled`** and **`authorizationMode`** properties in the `node.conf.yaml` or `server.conf.yaml` configuration files, or by using the **`mqsichangeauthmode`** command.

Procedure

Enable administration security for an integration node or integration server by following the steps in one of the following topics:

- “Enabling administration security by using the node.conf.yaml or server.conf.yaml configuration file” on page 2505
- “Enabling administration security by using the mqsichangeauthmode command” on page 2506

What to do next

Set the required permissions to enable users to complete the appropriate tasks on the integration node or integration server and resources. This task is described in "Authorizing users for administration" on page 2516. For more information about specifying authorization modes, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

Enabling administration security by using the node.conf.yaml or server.conf.yaml configuration file

Enable administration security on an integration node or integration server to control which users can complete specific tasks against it and its resources.

About this task

If you do not enable administration security for an integration node, all users are able to complete all actions against the integration node and all its managed integration servers. If you do not enable administration security for an independent integration server, all users are able to complete all actions against that integration server and its resources. If administration security is not enabled, web users can access the web user interface as the default user, with unrestricted access to data and resources.

You can enable administration security and specify the authorization mode for the integration node or server, by setting properties in the **`Admin Security`** section of the `node.conf.yaml` or `server.conf.yaml` configuration file for the relevant integration node or server.

Procedure

1. Open the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. In the relevant `node.conf.yaml` or `server.conf.yaml` file, set the **`basicAuth`**, **`authorizationEnabled`**, and **`authorizationMode`** properties:
   a) Enable basic authentication by setting the **`basicAuth`** property to `true`:

   ```
   basicAuth: true
   ```

   b) Enable authorization by setting the **`authorizationEnabled`** property to `true`:

   ```
   authorizationEnabled: true
   ```

   c) Specify the authorization mode that you require, by setting the **`authorizationMode`** property to `file`, `mq`, or `ldap`.

   For example, to enable administration security with the file-based authorization mode:

   ```
   authorizationMode: file
   ```

3. Optional: If basic authentication is enabled, you can use the following properties to control the maximum number of login attempts that can be made within a specified period before the client is locked out:
   - **authMaxAttempts** - the maximum number of login attempts that can be made during the specified period before being blocked (default is 5)
   - **authAttemptsDuration** - the period of time (in seconds) during which the maximum number of login attempts can be made before being blocked (default is 300)
   - **authBlockedDuration** - the period of time (in seconds) for which the client is blocked from logging in when the maximum number of login attempts has been reached without success (default is 300).

   For example:

```
Admin Security
   Authentication
   #  If basicAuth is enabled, a maximum of authMaxAttempts authentication attempts are
 allowed for a client within period authAttemptsDuration
   #  If authMaxAttempts is reached without success, the client is locked out for period
 authBlockedDuration
      basicAuth: true                # Clients web user name and password will be
 authenticated when set true
      authMaxAttempts: 3             # Max allowed authentication attempts
      authAttemptsDuration: 300      # Authentication attempts period in seconds
      authBlockedDuration: 600       # Authentication blocked period in seconds
```

4. Save the modified `.yaml` file.
5. Restart the integration node or server for the changes to take effect.

What to do next

Set the required permissions to enable users to complete the appropriate tasks on the integration node or server and its resources. This task is described in "Authorizing users for administration" on page 2516. For more information about specifying authorization modes, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

Enabling administration security by using the `mqsichangeauthmode` command

Enable administration security on an integration node or server to control which users can complete specific tasks against it and its resources.

About this task

If administration security is not enabled, web users can access the web user interface as the default user, with unrestricted access to data and resources.

If you do not enable administration security for an integration node, all users are able to complete all actions against the integration node, the integration servers that it manages, and all associated resources. If you do not enable administration security for an independent integration server, all users are able to complete all actions against the integration server and its resources.

You can enable administration security and specify the authorization mode for the integration node or server by using the **mqsichangeauthmode** command to specify queue-based administration security, or file-based authorization mode. For information about LDAP authorization, see "Configuring authorization by using LDAP groups" on page 2537.

You can set additional administration security properties, such as the maximum number of login attempts that can be made before a client is locked out, by editing the `node.conf.yaml` or `server.conf.yaml` configuration file for the integration node or server. For more information, see "Enabling administration security by using the node.conf.yaml or server.conf.yaml configuration file" on page 2505.

Procedure

To enable administration security by using the **mqsichangeauthmode** command, complete the following steps:

1. Stop the integration node by using the web user interface or by running the command.
2. Enable administration security and specify the authorization mode, by using the command:
   a) Enable administration security by specifying the **-s active** parameter on the **mqsichangeauthmode** command.
   b) Specify the authorization mode that you require by using the **-m** parameter.

      For example, to enable administration security with the file-based authorization mode for the ACE11NODE integration node, enter the following command:

      ```
      mqsichangeauthmode ACE11NODE -s active -m file
      ```

      where `-s active` enables administration security for the integration node, and `-m file` specifies the file-based authorization mode.

      If you have chosen to use queue-based administration security (`mq` mode), ensure that the queue manager specified on the integration node is running.
3. Ensure that the system user ID that runs the **mqsichangeauthmode** command is a member of the mqbrkrs group. Read, write, and execute permissions are granted automatically on the integration node to all user IDs that belong to this group.

   If you have chosen to use queue-based administration security (`mq` mode), ensure that the user ID is a member of the mqm group, with permission to create the required authorization queues. If the queues are not created automatically, you can create them manually; see "Creating the default system queues on an IBM MQ queue manager" on page 99. For more information, see "Authorization queues for queue-based administration security" on page 2501.

   Manage the membership of the mqbrkrs group and mqm group with care, and ensure that this level of authorization is granted only to users who require it.
4. Start the integration node by using the web user interface or the command.

What to do next

Set the required permissions to enable users to complete the appropriate tasks on the integration node and its resources. This task is described in "Authorizing users for administration" on page 2516. For more information about specifying authorization modes, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

Authenticating users for administration

Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. You can control access to the IBM App Connect Enterprise administration interfaces by using the authentication capabilities that are provided with the product.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Setting up administration security" on page 2504
- "Role-based security" on page 2503

About this task

IBM App Connect Enterprise provides authentication support for the following administration interfaces:

- IBM App Connect Enterprise web user interface.

- IBM App Connect Enterprise RESTful application programming interface (REST API).
- IBM App Connect Enterprise Toolkit.
- IBM App Connect Enterprise commands.

If administration security authentication (**basicAuth**) is enabled, users of the web user interface and the REST API must log in with a user ID and password. If LDAP authentication is enabled on the integration node or independent integration server, then all users are authenticated by the LDAP server. Any local passwords are ignored. If LDAP authentication is not configured, then the user ID and password are checked against the credentials that are held in the integration node or independent integration server. Users' access to data and resources is controlled by the permissions that are associated with their role. For more information, see “Role-based security” on page 2503.

If administration security is not enabled, web users can interact with the IBM App Connect Enterprise web user interface without logging on. They interact with the web UI as the 'default' user and can access all data and resources. For users of the REST API, all REST requests are unrestricted if administration security is not enabled.

For the following administration interfaces, authentication is provided only by the system login; no additional authentication is carried out:

- IBM App Connect Enterprise Toolkit
- IBM Integration API
- IBM App Connect Enterprise commands (when they make a local connection, specifying only the integration node name)

For more information about authenticating users for administration, see “Managing web user accounts” on page 2510 and “Accessing the web user interface” on page 89.

For more information about authenticating web user accounts by using LDAP, see “Enabling LDAP authentication” on page 2511.

For more information about authorizing users based on the role to which they are assigned, see “Authorizing users for administration” on page 2516.

You can enable authentication for users of IBM App Connect Enterprise administration interfaces, either by using the **mqsichangeauthmode** command, or by setting security properties in the appropriate `.yaml` configuration file for your integration node or server.

Procedure

Enable authentication by completing the steps in one of the following tasks:

- “Enable authentication by using the mqsichangeauthmode command” on page 2508
- “Enable authentication by modifying the node.conf.yaml or server.conf.yaml file” on page 2509

Enable authentication by using the `mqsichangeauthmode` command

About this task

Complete the following steps to enable authentication for users of the IBM App Connect Enterprise administration interfaces, by using the **mqsichangeauthmode** command:

Procedure

1. Run the **mqsichangeauthmode** command on your integration node or server. Specify the **-b** parameter to enable authentication only, as shown in the following example:

   ```
   mqsichangeauthmode -w myIntegrationServerWorkPath -b active
   ```

   In this example, authentication is enabled on the independent integration server, the work path of which is specified by the **-w** parameter.

   In the following example, authentication is enabled on the integration node ACE11NODE:

   ```
   mqsichangeauthmode ACE11NODE -b active
   ```

   Alternatively, you can specify the **-s** parameter, which configures both authentication and authorization. For more information, see command.
2. Restart your integration node or integration server for the changes to take effect.

Enable authentication by modifying the node.conf.yaml or server.conf.yaml file

About this task

Modify properties in the `.yaml` configuration file for your integration node or integration server to enable authentication for users of the IBM App Connect Enterprise administration interfaces.

Procedure

1. Open the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. In the Admin Security Authentication section of the `.yaml` configuration file, set the **basicAuth** property to `true`:

   ```
   basicAuth: true
   ```
3. Optional: If basic authentication is enabled, you can use the following properties to set the maximum number of login attempts that can be made within a specified period before the client is locked out:
   - **authMaxAttempts** - the maximum number of login attempts that can be made during the specified period before the user is blocked from logging in (default is 5)
   - **authAttemptsDuration** - the time (in seconds) during which the maximum number of login attempts can be made before the user is blocked from logging in (default is 300)
   - **authBlockedDuration** - the time (in seconds) for which the client is blocked from logging in when the maximum number of login attempts has been reached without success (default is 300).

   For example,

   ```
   Admin Security
      Authentication
      #  If basicAuth is enabled, a maximum of authMaxAttempts authentication attempts are
    allowed for a client within period authAttemptsDuration
      #  If authMaxAttempts is reached without success, the client is locked out for period
    authBlockedDuration
         basicAuth: true                                                  # Clients web user
    name and password will be authenticated when set true
         authMaxAttempts: 3                                               # Max allowed
    authentication attempts
         authAttemptsDuration: 300                                        # Authentication
    attempts period in seconds
   ```

```
      authBlockedDuration: 600                                          # Authentication
  blocked period in seconds
```

4. Save the `.yaml` configuration file.
5. Restart your integration node or integration server for the changes to take effect.

Managing web user accounts

IBM App Connect Enterprise administrators can use the **mqsiwebuseradmin** command to create a new web user, to set, or change a web user's password, to remove a web user or to assign a web user to a role.

Before you begin

Read the following topics:

- web user interface
- “Accessing the web user interface” on page 89
- “Role-based security” on page 2503

About this task

If administration security is enabled, web users can access the web user interface only when they log on using their web user account. As an administrator, you can create multiple roles, with different permissions assigned to them. You can then assign one or more web users to a role. The web users access to data and resources is controlled by the permissions that are set for their role. For more information, see “Role-based security” on page 2503.

If administration security is not enabled, web users can interact with the web user interface without logging on. They interact with the web user interface as the 'default' user and can access all data and resources.

If the integration node or integration server is configured to use file-based authorization, you assign permissions to the role by using the **mqsichangefileauth** command or in the node.conf.yaml or server.conf.yaml configuration file. Permissions are set for the integration node or integration server, and the data capture object. For more information about setting permissions for file-based authorization, see “Setting file-based permissions” on page 2530.

If the integration node is configured to use queue-based authorization (**mq** mode), you must create a system user ID on the operating system that is running your integration node. This system user is then used as a role, and you assign permissions to it by setting them on the following authorization queues:

- SYSTEM.BROKER.AUTH
- SYSTEM.BROKER.AUTH.*integrationServerName*
- SYSTEM.BROKER.DC.AUTH

For more information about setting permissions for queue-based authorization, see “Setting queue-based permissions” on page 2533.

If the integration node or integration server is configured to use LDAP authorization, you assign permissions to the role by configuring node.conf.yaml or server.conf.yaml configuration file. Permissions are set for the integration node or integration server. For information about setting permissions for LDAP authorization, see “Configuring authorization by using LDAP groups” on page 2537.

For more information about how to set the permissions that are required for using the web user interface, see “Controlling access to data and resources in the web user interface” on page 2544.

When you define your roles and set the required permissions, you can assign web users to the appropriate role, and they acquire permissions through their assigned role.

Procedure

Complete these steps to grant access to web users based on their assigned role:

1. Stop the integration node or integration server for which you are configuring administration security.
2. Enable administration security for the integration node or server by using the **mqsichangeauthmode** command, or in the node.conf.yaml or server.conf.yaml configuration file, specifying your chosen authorization mode.

   For example, to enable administration security with the file-based authorization mode for the ACE11NODE integration node, enter the following command.

   ```
   mqsichangeauthmode ACE11NODE -s active -m file
   ```

   where `-s active` enables administration security for the integration node, and `-m file` specifies the file-based authorization mode.

   The following example enables authentication only (not authorization) on an independent integration server whose work path is specified by the **-w** parameter:

   ```
   mqsichangeauthmode -w myIntegrationServerWorkpath -b active
   ```

   For more information, see command and “Enabling administration security” on page 2504.
3. Define the roles and their associated permissions. You can assign permissions to each role that you identify. For example, you might decide that your web users can be categorized into two main roles: web administrators and web users. Define a role for each of these groups of users (for example, `ACEUsers` and `ACEAdmins`) with permissions that allow them to complete the required tasks, such as viewing or modifying resources:

   - If the integration node or server is configured to use file-based authorization (**file** mode), you define the roles and associated permissions by using the **mqsichangefileauth** command or in the node.conf.yaml or server.conf.yaml configuration file. For more information about setting permissions for file-based authorization, see “Setting file-based permissions” on page 2530.
   - If the integration node is configured to use queue-based authorization (**mq** mode), you must create a system user ID on the operating system for each role that you identify. You must then assign permissions to the system user ID, which is then used as a role. For information about setting permissions for queue-based authorization, see “Setting queue-based permissions” on page 2533.
   - If the integration node or server is configured to use LDAP authorization (**ldap** mode), you define the roles and associated permissions by configuring the node.conf.yaml or server.conf.yaml configuration file. For information about setting permissions for LDAP authorization, see “Configuring authorization by using LDAP groups” on page 2537.

   For more information about setting the appropriate permissions, see “Authorizing users for administration” on page 2516 and “Controlling access to data and resources in the web user interface” on page 2544.
4. Use the **mqsiwebuseradmin** command to create your web user accounts and assign them to the appropriate roles.

   For more information, see command. For more information about roles, see “Role-based security” on page 2503. For more information about authenticating web user accounts by using LDAP, see “Enabling LDAP authentication by using the mqsichangeproperties command” on page 2515.

   If you add a local password by using the **-a** parameter, and LDAP authentication is enabled, the local password is ignored. When LDAP authentication is enabled, all web user logins must be authenticated by using LDAP. Any local passwords are ignored.
5. Restart the integration node or integration server for the changes to take effect.

Enabling LDAP authentication

Web user accounts can be authenticated against a Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS) server. You can authenticate web users by using the REST API, the web user

interface, the IBM App Connect Enterprise Toolkit, or custom integration applications that use the Integration API.

Before you begin

Ensure that you have an LDAP server that is LDAP Version 3 compliant, for example:

- IBM Tivoli Directory Server
- Microsoft Active Directory
- OpenLDAP

About this task

You can enable LDAP authentication for an integration node. See "Enabling an integration node to use LDAP for authentication" on page 2512. You can also enable LDAP authentication for an integration server. See "Enabling an integration server to use LDAP for authentication" on page 2513.

If a web user account has a local password, and LDAP authentication is enabled, the local password is ignored. When LDAP authentication is enabled, all web user logins must be authenticated by using LDAP. Any local passwords are ignored.

Enabling an integration node to use LDAP for authentication

About this task

You can enable LDAP authentication for an integration node by modifying the `node.conf.yaml` configuration file, by following the steps in the following procedure.
Alternatively, you can enable LDAP authentication for an integration node by running the `mqsichangeproperties` command. For more information, see "Enabling LDAP authentication by using the mqsichangeproperties command" on page 2515.

Procedure

1. Stop the integration node. See "Starting and stopping an integration node on Windows, Linux, and UNIX systems" on page 247.
2. Open the `node.conf.yaml` configuration file for your integration node by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters because they are invalid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
   The properties that you need to set are in the **`Admin Security`** section of the `node.conf.yaml` configuration file:

```
Admin Security:
#  Authentication
  #basicAuth: true                                                  # Clients web user name
 and password will be authenticated when set true
  #ldapUrl: ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]  # ldap authentication
 url
  #ldapBindDn: ldap::adminAuthentication                            # Resource alias or
 full bind dn
  #ldapBindPassword: ldap::adminAuthentication                      # Resource alias or
 bind password
  #  Authorization
  #authorizationEnabled: false     # Clients web user role will be authorized when set true
  #authorizationMode: 'file'       # Set authorization mode. Choose 1 of : ldap, file or mq
```

3. Set the following properties:
   a) Set the **basicAuth** property to `true`.
   b) Set the **ldapUrl** property to the URL of your LDAP server.
   c) Optional: Specify the service credentials to be used by the integration node when it binds to the LDAP server during a query. This information is required if the `sub` option is used in the **ldapUrl** and your LDAP server does not support anonymous bind.

      To specify these credentials, set the **ldapBindDn** and **ldapBindPassword** properties. For example:

      - **ldapBindDn**: *CN=XXXXXXXXX-XXXXXX,OU=PSST,OU=SAUsers,OU=PBS,OU=SS,DC=ent,DC=ds,DC=xxx,DC=xxx*
      - **ldapBindPassword**: *ldap::adminAuthentication*

      Set these properties by using one of the following methods:

      - Set the properties in the `node.conf.yaml` file (which stores the credential in clear text).
      - Set an alias name that you then use with the **mqsisetdbparms** command.
      - Set an alias name that you then use with the **mqsicredentials** command to store the LDAP bind DN and password.
4. Start the integration node. See "Starting and stopping an integration node on Windows, Linux, and UNIX systems" on page 247.
5. Optional: Configure the web user accounts for each user that you want to authorize. This step is not required if you intend to configure authorization by using LDAP groups. See "Configuring authorization by using LDAP groups" on page 2537. The step is required when you use file-based or queue-based authorization. See "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523. Either create new web user accounts or modify existing web user accounts:

   - Create a web user account by using the **mqsiwebuseradmin** command. For example,

     ```
     mqsiwebuseradmin intNode -c -u ldapusername -x -r sysrole
     ```

     where *ldapusername* is the username in the LDAP directory, and *sysrole* is the role to associate with the web user account. For more information about roles, see "Role-based security" on page 2503.
   - Modify an existing web user account to remove any local password. For example,

     ```
     mqsiwebuseradmin intNode -m -u ldapusername -x -r sysrole
     ```

     You can modify an existing web user account to be authenticated by using LDAP only if the existing username matches the username in the LDAP directory. If the usernames do not match, you must create a new web user account.

Enabling an integration server to use LDAP for authentication

Procedure

1. Stop the integration server. See "Stopping an integration server" on page 254.
2. Open the `server.conf.yaml` configuration file for your integration server by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters because they are invalid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
   The properties that you need to set are in the **Admin Security** section of the `server.conf.yaml` configuration file:

   ```
   Admin Security:
   #  Authentication
   ```

```
 #basicAuth: true                                                    # Clients web user name
and password will be authenticated when set true
 #ldapUrl: ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]  # ldap authentication
url
 #ldapBindDn: ldap::adminAuthentication                              # Resource alias or
full bind dn
 #ldapBindPassword: ldap::adminAuthentication                        # Resource alias or
bind password
 #  Authorization
 #authorizationEnabled: false     # Clients web user role will be authorized when set true
 #authorizationMode: 'file'       # Set authorization mode. Choose 1 of : ldap, file or mq
```

3. Set the following properties:
   a) Set the **basicAuth** property to `true`.
   b) Set the **ldapUrl** property to the URL of your LDAP server.
   c) Optional: Specify the service credentials to be used by the integration server when it binds to the LDAP server during a query. This information is required if the sub option is used in the **ldapUrl** and your LDAP server does not support anonymous bind.

      To specify these credentials, set the **ldapBindDn** and **ldapBindPassword** properties. For example:

      - **ldapBindDn**: *CN=XXXXXXXXX-XXXXXX,OU=PSST,OU=SAUsers,OU=PBS,OU=SS,DC=ent,DC=ds,DC=xxx,DC=xxx*
      - **ldapBindPassword**: *ldap::adminAuthentication*

      Set these properties by using one of the following methods: s

      - Set the properties in the `server.conf.yaml` file (which stores the credential in clear text).
      - Set an alias name that you then use with the **mqsisetdbparms** command.
      - Set an alias name that you then use with the **mqsicredentials** command to store the LDAP bind DN and password.
4. Start the integration server. See "Starting an integration server" on page 250.
5. Optional: Configure the web user accounts for each user that you want to authorize. This step is not required if you intend to configure authorization by using LDAP groups. See "Configuring authorization by using LDAP groups" on page 2537. The step is required when you use file-based or queue-based authorization. See "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523. Either create new web user accounts or modify existing web user accounts:

   - Create a web user account by using the **mqsiwebuseradmin** command. For example,

     ```
     mqsiwebuseradmin intNode -c -u ldapusername -x -r sysrole
     ```

     where *intNode* is the name of the integration node, *ldapusername* is the username in the LDAP directory, and *sysrole* is the role to associate with the web user account. For more information about roles, see "Role-based security" on page 2503.
   - Modify an existing web user account to remove any local password. For example,

     ```
     mqsiwebuseradmin intNode -m -u ldapusername -x -r sysrole
     ```

     You can modify an existing web user account to be authenticated by using LDAP only if the existing username matches the username in the LDAP directory. If the usernames do not match, you must create a new web user account.

What to do next

You might want to authorize users for administration. For more information, see "Authorizing users for administration" on page 2516.

*Enabling LDAP authentication by using the **mqsichangeproperties** command*
Web user accounts can be authenticated against a Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS) server. You can authenticate web users by using the REST API, the web user interface, the IBM App Connect Enterprise Toolkit, or custom integration applications that use the Integration API.

Before you begin

Ensure that you have an LDAP server that is LDAP Version 3 compliant, for example:

- IBM Tivoli Directory Server
- Microsoft Active Directory
- OpenLDAP

Procedure

1. Open a command window that is configured for your environment.
2. To set the LDAP server that you want to use for authentication, enter the following command on the command line.

```
mqsichangeproperties intNode -b RestAdminListener -n ldapUrl
-v "ldapURL"
```

where *intNode* is the name of your integration node and *ldapURL* is the URL for your LDAP.

Enter the *ldapURL* by using the following syntax:

```
ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]
```

ldap or ldaps
(Required) Fixed protocol string. Use `ldaps` to specify that SSL is used.

server
(Required) Name or IP address of the LDAP server.

port
(Optional) Port on the LDAP server. If SSL is not enabled, the default port is 389. If SSL is enabled, the default port is 636.

baseDN
(Required) String that defines the base distinguished name (DN) of all users in the directory. If users exist in different subtrees, specify a common subtree under which a search on the username uniquely resolves to the required user entry, and set the sub attribute.

If users who need access to the integration exist in multiple base DNs, you can specify more than one base DN in the *ldapURL* by enclosing each base DN in parentheses. The following syntax shows how to specify the *ldapURL* when users exist in three base DNs:

```
ldap[s]://server[:port]/|(baseDN1)(baseDN2)(baseDN3)[?[uid_attr][?[base|sub]]]
```

uid_attr
(Optional) String that defines the attribute to which the incoming username maps, typically uid, CN, or email address. The default is uid.

base or sub
(Optional) Defines whether to run a base or subtree search. If `base` is selected, the authentication is faster because the DN of the user is constructed from the uid_att, username, and baseDN

values. If sub is selected, a search must be completed before the DN can be resolved. The default is sub.

For example,

```
ldap://ldap.acme.com:389/ou=sales,o=acme.com
```

or

```
ldaps://localhost:636/ou=sales,o=acme?cn?base
```

Put public server certificates in the integration node truststore for use with LDAPS connections; do not put them in the RestAdminListener truststore.

3. Optional: If you want to configure authorization by using LDAP groups, you must set the **authorizationMode** property to `'ldap'`, by using the **mqsichangeauthmode** command, or by setting it in the `.yaml` configuration file.

   If you set the **authorizationMode** property to `'file'`, or `'mq'`, and you enabled LDAP authentication, you must configure the authorization role for web user account for each user that you want to authorize. To configure the role for each web user, either create new web user account or modify any existing web user accounts as follows:

   - Create a web user account by using the **mqsiwebuseradmin** command. For example,

     ```
     mqsiwebuseradmin intNode -c -u ldapusername -x -r sysrole
     ```

     where *ldapusername* is the username in the LDAP directory, and *sysrole* is the role to associate with the web user account. For more information about roles, see “Role-based security” on page 2503.

     If you add a local password by using the **-a** parameter, and LDAP authentication is enabled, the local password is ignored. When LDAP authentication is enabled, all web user logins must be authenticated by using LDAP. Any local passwords are ignored.

   - Modify an existing web user account to remove any local password. For example,

     ```
     mqsiwebuseradmin intNode -m -u ldapusername -x -r sysrole
     ```

     You can modify an existing web user account to be authenticated by using LDAP only if the existing username matches the username in the LDAP directory. If the usernames do not match, you must create a new web user account.

What to do next

You might want to authorize users for administration. For more information, see “Authorizing users for administration” on page 2516.

Authorizing users for administration

Authorize users to complete specific tasks against an integration node or server and its resources.

About this task

Three levels of permission are supported for IBM App Connect Enterprise administration security: read, write, and execute. These permissions can be applied to each role for the following types of objects:

- Integration node resources
- Integration server resources
- Data resources (record-replay)

For more information about roles, see “Role-based security” on page 2503.

You can enable administration security for an integration node or integration server, either by using the **mqsichangeauthmode** command, or by setting the security properties in the `node.conf.yaml` or `server.conf.yaml` configuration files.

If you enable administration security for an integration node, you can also choose which authorization mode (file-based, queue-based, or LDAP authorization) will be used for setting permissions. For independent integration servers (which are not managed by an integration node), you can specify file-based authorization or LDAP authorization.

You control access to an **integration node** (and the integration servers that it manages) by setting file-based or queue-based permissions, or LDAP authorization. You can set file-based permissions either by using the **mqsichangefileauth** command, or by setting properties in the `node.conf.yaml` configuration file. You can set queue-based permissions by using IBM MQ authorization queues on the queue manager that is specified on the integration node. You can set LDAP authorization by setting properties in the `node.conf.yaml` configuration file. For information about the permissions that are required for working with an integration node and its resources, see "Permissions for acting on integration nodes, integration servers, and resources" on page 2518.

When you enable administration security for an **integration node**, the default mode of authorization depends on whether a queue manager is specified on the integration node. If a queue manager has been specified, authorization for the integration node is based on IBM MQ queues by default (`mq` mode), and the required queues used for setting authorization are created automatically when the integration node is created. If you create an integration node without specifying an associated queue manager, file-based authorization (`file` mode) is used by default.

You can control access to an independent **integration server** (which is not managed by an integration node) by using either file-based permissions, or LDAP authorization. You can set file-based permissions either by using the **mqsichangefileauth** command or by setting properties in the `server.conf.yaml` configuration file. You can set LDAP authorization by setting properties in the `server.conf.yaml` configuration file.

If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, you must set the required permissions that enable the connection to be made to the queue manager that is specified on the integration node. For information about these permissions, see "Permissions for connecting to a queue manager" on page 2521. When you have set the required permissions for connecting to the queue manager, you can set the permissions that authorize users to act on the integration node and its resources.

For information about authentication, see "Authenticating users for administration" on page 2507.

Procedure

Complete the following steps to set the required authorization mode and to authorize users to work with an integration node or server and its resources:

1. Ensure that administration security for the integration node or server is enabled and configured to use the required authorization mode, as described in "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523.

   To find out which authorization mode is currently in effect, see "Checking the authorization mode" on page 2529.
2. If you are using queue-based administration security for an integration node, set the required permissions to enable users to connect to the IBM MQ queue manager.

   For information about these permissions, see "Permissions for connecting to a queue manager" on page 2521.
3. Set the required permissions to enable users to complete tasks on an integration node or server and its resources. For information about the permissions that are required, see "Permissions for acting on integration nodes, integration servers, and resources" on page 2518.

   For information about how to set the permissions, see "Setting file-based permissions" on page 2530, "Setting queue-based permissions" on page 2533, and "Configuring authorization by using LDAP groups" on page 2537.

4. You can control web users' access to data and resources by assigning permissions to the role that the users are assigned to. For more information, see “Controlling access to data and resources in the web user interface” on page 2544.

Permissions for acting on integration nodes, integration servers, and resources

Permissions are required for users to act on integration nodes, integration servers, and resources.

Authorization to perform administrative tasks is determined by the permissions that are granted to the role to which the user has been assigned. The following tables show the permissions that are required for the user's assigned role in order for them to carry out specific tasks, depending on whether you are using queue-based, file-based, or LDAP administration security. For information about how to specify an authorization mode, see “Configuring administration security to use file-based, queue-based, or LDAP authorization” on page 2523.

If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, you must also set the required permissions for connecting to the queue manager that is specified on the integration node. For information about the permissions that are required for connecting to the queue manager, see “Permissions for connecting to a queue manager” on page 2521.

Table 78. MQ queue-based permissions required for acting on an integration node

Action	Integration node permission	MQ queue-based security: IBM MQ queue	MQ queue-based security: IBM MQ permission (set on setmqaut command)
View	read	SYSTEM.BROKER.AUTH	+INQ
Create	write	SYSTEM.BROKER.AUTH	+PUT
Delete	write	SYSTEM.BROKER.AUTH	+PUT
Modify	write	SYSTEM.BROKER.AUTH	+PUT
Start	execute	SYSTEM.BROKER.AUTH	+SET
Stop	execute	SYSTEM.BROKER.AUTH	+SET
Inject	execute	SYSTEM.BROKER.AUTH	+SET

Table 79. File-based permissions required for acting on an integration node

Action	Integration node permission	File-based security: Object flag (set on mqsichangefileauth command)	File-based security: File permission (set on mqsichangefileauth command or in a node.conf.yaml file)
View	read		read+
Create	write		write+
Delete	write		write+
Modify	write		write+
Start	execute		execute+
Stop	execute		execute+
Inject	execute		execute+

Where no object flag is specified on the command command, the file-based permissions are set at the level of the integration node.

Table 80. LDAP permissions required for acting on an integration node

Action	Integration node permission	LDAP security: Permission (set in node.conf.yaml configuration file)
View	read	'read+'
Create	write	write+
Delete	write	'write+'
Modify	write	'write+'
Start	execute	'execute+'
Stop	execute	'execute+'
Inject	execute	'execute+'

Table 81. MQ queue-based permissions required for acting on an integration server that is managed by an integration node

Action	Integration node permission	MQ queue-based security: IBM MQ queue	MQ queue-based security: IBM MQ permission (set on setmqaut command)
View	read	SYSTEM.BROKER.AUTH.*integrationServerName*	-INQ
Create	write	SYSTEM.BROKER.AUTH.*integrationServerName*	-PUT
Delete	write	SYSTEM.BROKER.AUTH.*integrationServerName*	-PUT
Modify	write	SYSTEM.BROKER.AUTH.*integrationServerName*	-PUT
Start	execute	SYSTEM.BROKER.AUTH.*integrationServerName*	-SET
Stop	execute	SYSTEM.BROKER.AUTH.*integrationServerName*	-SET

Table 82. File-based permissions required for acting on an integration server that is managed by an integration node

Action	Integration node permission	File-based security: Object flag (set on mqsichangefileauth command)	File-based security: File permission (set on mqsichangefileauth command or in a node.conf.yaml file)
View	read	-e *integration_server*	read+
Create	write	-e *integration_server*	write+
Delete	write	-e *integration_server*	write+
Modify	write	-e *integration_server*	write+
Start	execute	-e *integration_server*	execute+
Stop	execute	-e *integration_server*	execute+

Table 83. LDAP permissions required for acting on an integration server that is managed by an integration node

Action	Integration node permission	LDAP security: Permission (set in the node.conf.yaml configuration file of the integration node that manages the integration server)
View	read	'read+'
Create	write	write+
Delete	write	'write+'
Modify	write	'write+'
Start	execute	'execute+'
Stop	execute	'execute+'

Table 84. File-based permissions required for acting on an independent integration server (not managed by an integration node)

Action	File-based security: Object flag (set on mqsichangefileauth command)	File-based security: File permission (set on mqsichangefileauth command or in a server.conf.yaml file)
View	-e *integration_server*	read+
Create	-e *integration_server*	write+
Delete	-e *integration_server*	write+
Modify	-e *integration_server*	write+
Start	-e *integration_server*	execute+
Stop	-e *integration_server*	execute+
Inject	-e *integration_server*	execute+

Table 85. LDAP permissions required for acting on an independent integration server (not managed by an integration node)

Action	LDAP security: Permission (set in server.conf.yaml configuration file)
View	'read+'
Create	'write+'
Delete	'write+'
Modify	'write+'
Start	'execute+'
Stop	'execute+'
Inject	'execute+'

Table 86. Permissions required for acting on a data object					
Action	**Integration node permission**	**MQ queue-based security: IBM MQ queue**	**MQ queue-based security: IBM MQ permission (set on setmqaut command)**	**File-based security: Object flag (set on mqsichangefileauth command)**	**File-based security: File permission (set on mqsichangefileauth command)**
View	read	SYSTEM.BROKER.DC.AUTH	+INQ	-o *Data*	read+
	read	SYSTEM.BROKER.DC.AUTH.*integrationServerName*	+INQ	-o *Data*	read+
Replay	execute	SYSTEM.BROKER.DC.AUTH.*integrationServerName*	+SET	-o *IntegrationServerName* -o *Data*	execute+

File-based and LDAP-based permissions:

When you create an integration server on an integration node for which either file-based or LDAP security has been enabled, you must explicitly set the permissions for the integration server and then restart the integration node for the changes to take effect. For more information, see "Configuring authorization for an integration node by modifying the node.conf.yaml file" on page 2523.

Queue-based permissions:

If the queue-based mode of administration security (*mq* mode) is enabled when you create an integration node, the queue SYSTEM.BROKER.AUTH is created. Read, write, and execute permissions are granted automatically to the user group mqbrkrs on this queue. The SYSTEM.BROKER.AUTH queue is created as a local queue, and is used to define which users are authorized to perform actions on the integration node and the integration node properties.

When you create an integration server on an integration node for which you have enabled queue-based security, the integration server authorization queue SYSTEM.BROKER.AUTH.*integrationServerName* is created, where *integrationServerName* is the name of the integration server. Read, write, and execute permissions are automatically granted to the user group mqbrkrs on this queue.

When you use the **mqsicreatebroker** command to create an integration node with an associated queue manager, the SYSTEM.BROKER.DC.AUTH queue is created automatically. If you create an integration node without specifying a queue manager, you can modify the integration node afterwards to specify a queue manager and enable administration security in mq mode; however, you must also create the SYSTEM.BROKER.DC.AUTH queue. For information about creating the system queues, see "Creating the default system queues on an IBM MQ queue manager" on page 99.

For more information about the creation of authorization queues, see "Authorization queues for queue-based administration security" on page 2501.

Permissions for connecting to a queue manager

If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, permissions are required for connecting to a queue manager, in addition to the specific permissions that are required for acting on integration nodes and resources.

If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, you must set the required permissions that enable the connection to be made to the queue manager that is specified on the integration node.

The following topics summarize the IBM MQ permissions that are required to connect to the queue manager that is specified on the integration node:

- "Permissions for administrators" on page 2522
- "Permissions for users connecting to the integration node" on page 2522

When you have set the required permissions for connecting to the queue manager, you must also set the permissions that are required to enable users to work with the integration node and its resources. For more information about these permissions, see "Permissions for acting on integration nodes, integration servers, and resources" on page 2518. You can use either queue-based or file-based permissions, or LDAP authorization to authorize users to access resources; for information about how to select the authorization mode and set the permissions, see the following topics:

- "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523
- "Setting file-based permissions" on page 2530
- "Setting queue-based permissions" on page 2533
- "Configuring authorization by using LDAP groups" on page 2537

Permissions for administrators
If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, additional permissions are required by administrators.

About this task

When you activate administration security with the authorization mode set to `mq`, the IBM MQ permissions for inquire, put, and set are granted for the group mqbrkrs for the queue SYSTEM.BROKER.AUTH. These permissions grant read, write, and execute authority on the integration node and its properties to all system user IDs that are members of mqbrkrs, where the system ID is provided by a local command or Toolkit, or corresponds to the configured role for a remote or web user.

If you want additional user IDs to have administrator authorization, either add those IDs to the group mqbrkrs, or add IBM MQ permissions for inquire, put, and set for those user IDs to this queue.

The following table summarizes the IBM MQ permissions that are required in order to connect to the queue manager that is specified on the integration node:

Object	Name	Permissions
Queue manager	The queue manager associated with the integration node; for example, ACEQMGR	Connect Inquire

For information about the permissions that are required to enable users to act on IBM App Connect Enterprise resources, see "Permissions for acting on integration nodes, integration servers, and resources" on page 2518.

Permissions for users connecting to the integration node
If you are using any IBM App Connect Enterprise functions that require access to IBM MQ, additional permissions are required to enable users to connect to the integration node.

About this task

If a user or application wants to connect to an integration node, you must grant them the appropriate permissions. All applications written to the IBM Integration API, and users of the IBM App Connect Enterprise Toolkit, require permissions based on their expected actions. The following table shows the IBM MQ permissions that are required in order to connect to the queue manager that is specified on the integration node:

Object	Name	Permissions
Queue manager	The queue manager associated with the integration node; for example, ACEQMGR	Connect Inquire

Notes:

1. Users and applications can connect to the integration node without this level of authority, but are unable to request actions against the integration node, including viewing properties.

For information about the permissions that are required to enable users to act on IBM App Connect Enterprise resources, see "Permissions for acting on integration nodes, integration servers, and resources" on page 2518.

Configuring administration security to use file-based, queue-based, or LDAP authorization

You can enable administration security for an integration node or integration server, either by using the **mqsichangeauthmode** command, or by setting the security properties in the `node.conf.yaml` or `server.conf.yaml` configuration file. If you enable administration security for an integration node, you can also choose which authorization mode (file-based, queue-based, or LDAP authorization) will be used for setting permissions. For independent integration servers (which are not managed by an integration node), you can specify file-based authorization, or LDAP authorization.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Authorizing users for administration" on page 2516

About this task

You control access to an integration node (and the integration servers that it manages) by setting file-based, or queue-based permissions, or LDAP authorization. You can set file-based permissions either by using the **mqsichangefileauth** command, or by setting properties in the `node.conf.yaml` configuration file. You can set queue-based permissions by using IBM MQ authorization queues on the queue manager that is specified on the integration node. You can set LDAP authorization by setting properties in the `node.conf.yaml` configuration file.

You can control access to an independent integration server (which is not managed by an integration node) by using file-based permissions or LDAP authorization. You can set file-based permissions either by using the **mqsichangefileauth** command or by setting properties in the `server.conf.yaml` configuration file. You can set LDAP authorization by setting properties in the `server.conf.yaml` configuration file.

To configure authorization for an integration node or integration server, complete the steps in one of the following topics:

- "Configuring authorization for an integration node by modifying the node.conf.yaml file" on page 2523
- "Configuring authorization for an integration server by modifying the server.conf.yaml file" on page 2526
- "Configuring authorization by using the mqsichangeauthmode command" on page 2528
- "Configuring authorization by using LDAP groups" on page 2537

Configuring authorization for an integration node by modifying the node.conf.yaml file

You can configure integration nodes (and the integration servers that they manage) to use either file-based authorization, queue-based authorization, or LDAP authorization, by setting the security properties in the `node.conf.yaml` configuration file.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Authorizing users for administration" on page 2516
- "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523

About this task

Authorization to perform administrative tasks is determined by the permissions that are granted to the role to which the web user has been assigned. When using LDAP authorization, a web user's role is determined by mapping the LDAP groups that they belong to with a predefined role, as described in "Configuring authorization by using LDAP groups" on page 2537. You can then set file-based permissions for that role. For more information about controlling access by using file-based permissions, see "Setting file-based permissions" on page 2530. For information about controlling access by using queue-based permissions, see "Setting queue-based permissions" on page 2533.

When you enable administration security for an integration node, the default mode of authorization depends on whether a queue manager is specified on the integration node. If a queue manager has been specified, administration security for the integration node is based on IBM MQ queues by default (`mq` mode), and the required queues used for setting authorization are created automatically when the integration node is created. If you create an integration node without specifying an associated queue manager, file-based administration security is used by default (`file` mode).

For information about controlling access to an independent integration server, see "Configuring authorization for an integration server by modifying the server.conf.yaml file" on page 2526.

You configure authorization for an integration node (and its managed integration servers) by setting properties in the Security section of the integration node's `node.conf.yaml` file:

```
# Admin Security
  #  Authentication
  #  If basicAuth enabled, a maximum of authMaxAttempts authentication attempts are allowed for
 a client within period authAttemptsDuration
  #  If authMaxAttempts is reached without success, the client is locked out for period
 authBlockedDuration
  #basicAuth: false                                                    # Clients web user name
 and password will be authenticated when set true
  webUserPasswordHashAlgorithm: PBKDF2-SHA-512                        # Algorithm used to hash
 the password for webuser accounts.
  #authMaxAttempts: 5                                                 # Max allowed
 authentication attempts
  #authAttemptsDuration: 300                                          # Authentication attempts
 period in seconds
  #authBlockedDuration: 300                                           # Authentication blocked
 period in seconds
  #sessionTimeout: 86400                                              # Set the timeout in
 seconds that REST API/Web UI sessions can be inactive before becoming invalid. Default is 86400
 seconds (24 hours).
  #ldapUrl: ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]  # ldap authentication url
  #ldapBindDn: ldap::adminAuthentication                              # Resource alias or full
 bind dn
  #ldapBindPassword: ldap::adminAuthentication                        # Resource alias or bind
 password
  #  Authorization
  #authorizationEnabled: false     # Clients web user role will be authorized when set true
  #authorizationMode: 'file'       # Set authorization mode. Choose 1 of : ldap, file or mq
  #ldapAuthorizeUrl: ldap[s]://server[:port]/baseDN[?[attr_name][?[base|sub]][?filter_expr]]  #
 ldap authorization url
  #ldapCheckServerIdentity : true  # Disables hostname verification of ldaps server when set to
 false

Security:
  #dynamicUpdate: false            # If true, changes to security settings will be picked up
 automatically by the Integration Node, after a few seconds.
  LdapAuthorizeAttributeToRoleMap:
    # When 'authorizationMode' is ldap, set the mapping from a matched LDAP authorization
 attribute, as
    # configured in 'ldapAuthorizeUrl' to the ACE web user role name
    # e.g. map the following LDAP group DNs to web user roles 'adminRole', 'viewRole'
    #'cn=admins,cn=group,ou=ace': 'adminRole'
    #'cn=monitors,cn=group,ou=ace': 'viewRole'
  Node:
    Permissions:
      # Set Admin Security Authorization file permissions for the Integration Node by web user
 role using 'read+:write+:execute+' , or 'all+'
      # '+' grants permission, '-' denies permission
      # e.g. define the following web user roles 'viewRole' and 'adminRole'
      #viewRole:  'read+:write-:execute-'
      #adminRole: 'all+'
    DataPermissions:
```

```
      # Set Admin Security Authorization file permissions for Record and Replay web user roles
 using 'read+:write+:execute+' , or 'all+'
      # '+' grants permission, '-' denies permission. Record and Replay roles also require 'read
+' permission to be defined for the
      # Integration Node in the Permissions section above.
      # e.g. define the following web user roles 'dataViewer', 'dataReplayer' and 'adminRole'
      #dataViewer:  'read+:write-:execute-'
      #dataReplayer: 'read+:write-:execute-'
      #adminRole: 'all+'
  Server:
    # Set Admin Security Authorization file permissions for each named Integration Server
    # e.g. define the following web user roles 'viewRole' and 'adminRole' for Integration Server
 'server01'
    #server01:
      #Permissions:
        #viewRole:  'read+:write-:execute-'
        #adminRole: 'all+'
      #DataPermissions:
        # Set Admin Security Authorization file permissions for Record and Replay web user role
 using 'read+:write+:execute+' , or 'all+'
        # '+' grants permission, '-' denies permission.
        # e.g. define the following web user roles 'dataViewer', 'dataReplayer' and 'adminRole'
        #dataViewer:  'read+:write-:execute-'
        #dataReplayer: 'read+:write-:execute+'
        #adminRole: 'all+'
```

You set permissions in the `node.conf.yaml` file for the integration node and for all the integration servers that it manages. When you have added or modified permissions, ensure that you restart the integration node for the changes to take effect.

Procedure

Configure the authorization mode for an integration node by completing the following steps:

1. Open the `node.conf.yaml` configuration file for your integration node, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. If administration security has not already been enabled, enable it now by setting the **authorizationEnabled** property in the Admin Security section of the `node.conf.yaml` file to `true`:

   ```
   authorizationEnabled: true
   ```
3. Specify the authorization mode that you require, by setting the **authorizationMode** property to either `file`, `mq`, or `ldap`.

   For example, to configure the integration node to use file-based authorization, set the following value:

   ```
   authorizationMode: file
   ```
4. Assign permissions to the defined roles, by specifying a combination of `read`, `write`, and `execute`, or `all`, with + to grant permission, and - to deny permission; for example:

   ```
   viewRole:  'read+:write-:execute-'
   adminRole: 'all+'
   ```

   where the `viewRole` has only read permission, and the `adminRole` has permission for all actions.
5. Save the modified `node.conf.yaml` file.
6. Restart the integration node for the changes to take effect, by using the command.

What to do next

For information about specifying the authorization mode for an integration node by using the **mqsichangeauthmode** (instead of modifying the `node.conf.yaml` file), see “Configuring authorization by using the mqsichangeauthmode command” on page 2528. You can then set permissions as described

in “Setting file-based permissions” on page 2530, “Setting queue-based permissions on Linux, AIX, and Windows systems” on page 2534or “Setting file-based permissions” on page 2530.

Configuring authorization for an integration server by modifying the server.conf.yaml file
You can configure integration servers to use file-based authorization, or LDAP authorization, by setting the security properties in the `server.conf.yaml` configuration file.

Before you begin

Read the following topics:

- “Administration security overview” on page 2497
- “Authorizing users for administration” on page 2516

About this task

You can configure authorization for an independent integration server (which is not associated with an integration node) by setting file-based permissions, or LDAP authorization in the Security section of the integration server's `server.conf.yaml` configuration file:

```
# Admin Security
  #  Authentication
  #  If basicAuth enabled, a maximum of authMaxAttempts authentication attempts are allowed for
 a client within period authAttemptsDuration
  #  If authMaxAttempts is reached without success, the client is locked out for period
 authBlockedDuration
  #basicAuth: false                                                    # Clients web user name
 and password will be authenticated when set true
  webUserPasswordHashAlgorithm: PBKDF2-SHA-512                        # Algorithm used to hash
 the password for webuser accounts.
  #authMaxAttempts: 5                                                 # Max allowed
 authentication attempts
  #authAttemptsDuration: 300                                          # Authentication attempts
 period in seconds
  #authBlockedDuration: 300                                           # Authentication blocked
 period in seconds
  #sessionTimeout: -1                                                 # Client-side expiration
 time in seconds for REST API/Web UI sessions.
                                                                      # Negative values or zero
 signify that the session remain active
                                                                      # for the lifetime of the
 browser session. Defaults to -1.
  #serverSessionTimeout: 86400                                        # Server-side expiration
 time in secods for REST API/Web UI sessions. Can
                                                                      # be specified
 independently of the client-side timeout, allowing sessions
                                                                      # to be invalidated on the
 server before they are expired by the client. This
                                                                      # useful in particular
 when the client-side session lifetime is set to that
                                                                      # of the browser session
 ('BROWSER_EXIT' special
  #serverSessionTimeoutCheckInterval: 3600
  #ldapUrl: ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]  # ldap authentication url
  #ldapBindDn: ldap::adminAuthentication                              # Resource alias or full
 bind dn
  #ldapBindPassword: ldap::adminAuthentication                        # Resource alias or bind
 password
  #  Authorization
  #authorizationEnabled: false     # Clients web user role will be authorized when set true
  #authorizationMode: 'file'       # Set authorization mode. Choose 1 of : ldap, file
  #ldapAuthorizeUrl: ldap[s]://server[:port]/baseDN[?[attr_name][?[base|sub]][?filter_expr]]  #
 ldap authorization search url
  #ldapCheckServerIdentity : true  # Disables hostname verification of ldaps server when set to
 false

Security:
  LdapAuthorizeAttributeToRoleMap:
    # When 'authorizationMode' is ldap, set the mapping from a matched LDAP authorization
 attribute, as
    # configured in 'ldapAuthorizeUrl' to the ACE web user role name
    # e.g. map the following LDAP group DNs to web user roles 'adminRole', 'viewRole'
    #'cn=admins,cn=group,ou=ace': 'adminRole'
```

```
    #'cn=monitors,cn=group,ou=ace': 'viewRole'
  Permissions:
    # Set Admin Security Authorization file permissions by web user role using 'read+:write
+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission
    # e.g. define the following web user roles 'viewRole' and 'adminRole'
    #viewRole:  'read+:write-:execute-'
    #adminRole: 'all+'
  DataPermissions:
    # Set Admin Security Authorization file permissions for Record and Replay web user role
 using 'read+:write+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission.  Record and Replay roles also require 'read
+' permission to be defined
    # in the Permissions section above.
    # e.g. define the following web user roles 'dataViewer', 'dataReplayer' and 'adminRole'
    #dataViewer:  'read+:write-:execute-'
    #dataReplayer: 'read+:write-:execute+'
    #adminRole: 'all+'
```

For information about controlling access to an integration node, see “Configuring authorization for an integration node by modifying the node.conf.yaml file” on page 2523.

Procedure

Configure the authorization mode for an integration server by completing the following steps:

1. Open the `server.conf.yaml` configuration file for your integration server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. If administration security has not already been enabled, enable it now by setting the **authorizationEnabled** property in the `server.conf.yaml` file to `true`:

   ```
   authorizationEnabled: true
   ```
3. Specify the authorization mode that you require, by setting the **authorizationMode** property to either `file`, or `ldap`.

   For example, to configure the integration node to use file-based authorization, set the following value:

   ```
   authorizationMode: file
   ```
4. Assign permissions to the defined roles, by specifying a combination of `read`, `write`, and `execute`, or `all`, with + to grant permission, and - to deny permission; for example:

   ```
   viewRole:  'read+:write-:execute-'
   adminRole: 'all+'
   ```

   where the `viewOnly` role has only read permission, and the `admin` role has permission for all actions.
5. Save the modified `server.conf.yaml` file.
6. Restart the integration server for the changes to take effect.

What to do next

You can also configure authorization for an integration server by using the **mqsichangeauthmode** command, and then set permissions by using the **mqsichangefileauth** command. For more information, see “Configuring authorization by using the mqsichangeauthmode command” on page 2528 and “Setting file-based permissions” on page 2530.

*Configuring authorization by using the **mqsichangeauthmode** command*
Use the **mqsichangeauthmode** command to enable administration security for an integration node or integration server.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Authorizing users for administration" on page 2516

About this task

You can use the **mqsichangeauthmode** command to enable administration security for an integration node or integration server, and to configure the authorization mode to be used. If you are configuring administration security for an integration node, you can use the **mqsichangeauthmode** command to configure it to use either a file-based or queue-based mode of authorization. If you are configuring administration security for an independent integration server (which is not managed by an integration node), you can use the **mqsichangeauthmode** command to configure it to use file-based authorization only.

If you want to configure an integration node, or an independent integration server (which is not managed by an integration node) to use LDAP authorization, you can do this only by setting properties in the node.conf.yaml configuration file for the integration node or integration server. For more information, see "Configuring authorization for an integration node by modifying the node.conf.yaml file" on page 2523, and "Configuring authorization for an integration server by modifying the server.conf.yaml file" on page 2526.

Procedure

1. Use the **mqsichangeauthmode** command to enable administration security:
   a) Enable administration security by specifying either **-s active** to enable both authentication and authorization, or **-b active** to enable authentication only.
   For example:

   ```
   mqsichangeauthmode -w myIntegrationServerWorkpath -b active
   ```

   This example enables authentication on an independent integration server, whose work path is specified by the **-w** parameter.
   b) Optional: If you enable both authentication and authorization by setting **-s active**, you must also specify the required authorization mode by using the **-m** parameter:
   - For an integration node:
     - Specify **-m file** to use file-based permissions, which are set using the **mqsichangefileauth** command. If you create an integration node without specifying an associated queue manager, file-based administration security is used by default for the integration node.
     - Specify **-m mq** to use IBM MQ queues for setting permissions. You can use queue-based security only if you have installed IBM MQ and specified a queue manager on the integration node. If a queue manager is specified on the integration node, administration security is

queue-based by default, and the required queues used for setting authorization are created automatically when the integration node is created.

For example:

```
mqsichangeauthmode ACE11NODE -s active -m file
```

- For an integration server:
  - Specify **-m file** to use file-based permissions, which are set using the **mqsichangefileauth** command.

For more information, see command and command.

2. Restart the integration node for the changes to take effect.

What to do next

If you want to check which security mode is currently in effect, you can either check the settings in the `node.conf.yaml` or `server.conf.yaml` configuration files, or you can run the command, as described in "Checking the authorization mode" on page 2529.

Checking the authorization mode

Use the **mqsireportauthmode** command to display the authorization mode that is currently in effect for the integration node or integration server.

Before you begin

Read the following topics:

- "Authorizing users for administration" on page 2516
- "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523

About this task

Administration authority can be controlled by using IBM MQ queues that are owned by a queue manager specified on the integration node, or by file-based permissions that are set using the **mqsichangefileauth** command, or by LDAP authorization that is set by editing the security properties in the `node.conf.yaml` or `server.conf.yaml`configuration file. You can use the **mqsireportauthmode** command to display the administration security status and authorization mode that are currently in effect for the integration node or server.

Procedure

1. Run the **mqsireportauthmode** command to display the current administration security status and authorization mode.
   For example, display the current settings for an integration node called ACE11NODE:

   ```
   mqsireportauthmode ACE11NODE
   ```

   Alternatively, display the current administration security settings for an independent integration server, whose work path is specified by the **-w** parameter:

   ```
   mqsireportauthmode -w myIntegrationServerWorkpath
   ```

2. The output from the command shows whether administration security is active or inactive, and whether the current authorization mode is `file` (file-based), `mq` (queue-based), or ldap (LDAP authorization).

Setting file-based permissions

Grant and revoke administration authority by configuring file-based permissions for working with an integration node and its resources or and integration server and its resources.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Authorizing users for administration" on page 2516
- "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523

About this task

You can grant and revoke administration authority for an integration node and its managed integration servers, or for an independent integration server, by configuring file-based permissions for specified roles. You can configure these permissions by using the command, or by setting properties in the `node.conf.yaml` file (for an integration node and its managed integration servers) or in the `server.conf.yaml` file (for an independent integration server).

You can use file-based permissions for authorization if the file-based or LDAP-based mode of administration security has been specified for the integration node or server. For LDAP authorization, you must associate a role with the LDAP groups to which the user belongs, and then set file-based permissions for that role. For more information, see "Configuring authorization by using LDAP groups" on page 2537.

To specify an authorization mode for an integration node (and its managed integration servers) or an independent integration server, you can either use the command or set the **`authorizationEnabled`** and **`authorizationMode`** properties in the `node.conf.yaml` or `server.conf.yaml` configuration file.

Three levels of authorization permissions are supported for IBM App Connect Enterprise administration security: read, write, and execute. You can assign permissions to a role by specifying the type of permission followed by a plus (+) to grant permissions, or a minus (-) to revoke permissions:

- *read+/-*
- *write+/-*
- *execute+/-*
- *all+/-*

These permissions can be applied to each role for the following types of objects:

- Integration node resources
- Integration server resources
- Data objects (record-replay and business transaction monitoring)

Note: If you grant permissions to a role at the integration node level, those permissions are not applied to the integration servers that are managed by the integration node; you must also set permissions explicitly for individual integration servers.

Set file-based permissions for your integration node or integration server by completing one of the following tasks:

- "Setting permissions in the node.conf.yaml configuration file or server.conf.yaml configuration file" on page 2531
- "Setting permissions by using the mqsichangefileauth command" on page 2531

Setting permissions in the node.conf.yaml configuration file or server.conf.yaml configuration file

About this task

You can configure authorization for an integration node (and its managed integration servers) or for an independent integration server, by setting permissions for roles in the Security section of the `node.conf.yaml` or `server.conf.yaml` configuration file.

You can define a role and grant or deny permissions to it, by specifying them in the appropriate `.conf.yaml` file. In the following example, the role called `viewRole` has been granted read permission only, and the role called `adminRole` has been granted permission for all actions.

```
Permissions:
    viewRole:  'read+:write-:execute-'
    adminRole: 'all+'
```

Procedure

Configure the authorization mode for an integration node or server by completing the following steps:

1. Open the `node.conf.yaml` file (for an integration node and its managed integration servers) or the `server.conf.yaml` file (for an independent integration server), by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. If administration security has not already been enabled, activate authentication and authorization by setting the **basicAuth** and **authorizationEnabled** properties to `true`:

   ```
   basicAuth: true
   authorizationEnabled: true
   ```
3. Specify the file-based authorization mode, by setting the **authorizationMode** property to `file`:

   ```
   authorizationMode: file
   ```
4. Define the role and set its permissions, by specifying a combination of `read`, `write`, and `execute`, or `all`, with + to grant permission, and - to deny permission; for example:

   ```
   viewOnly:  'read+:write-:execute-'
   aceAdmin:     'all+'
   ```

   where the `viewOnly` role has only read permission, and the `aceAdmin` role has permission for all actions.
5. Save the modified file.
6. Restart the integration node or independent integration server, as appropriate, for the changes to take effect.

*Setting permissions by using the **mqsichangefileauth** command*

About this task

You specify the permissions as a comma-separated list of values. A value can be specified for each permission (*read*, *write*, and *execute*) only once in the list of values. For example, you cannot specify *all-,read+* because it would be attempting to set the *read* permission twice (once explicitly, and once as part of *all*). If *all* is specified, it must be the only value. If you specify *all-*, all permission records in the registry are removed.

Procedure

Follow these steps to set permissions for a role:

1. Ensure that administration security has been enabled for the integration node.

   For more information, see “Enabling administration security” on page 2504.
2. Use the **mqsichangefileauth** command to change the permissions that are assigned to a role. For example, the following command shows how to set permissions on an integration server that is associated with an integration node:

   ```
   mqsichangefileauth ACE11NODE -r aceAdmins -e default -p read+,execute+
   ```

   In this example, the role `aceAdmins` is granted execute and read permission on `ACE11NODE.default` (the `default` integration server on the `ACE11NODE` integration node). If this role did not previously exist, the write permission is disabled.

   The following example shows how to set permissions on an independent integration server (which is not managed by an integration node):

   ```
   mqsichangefileauth -w server_workpath -r aceAdmins -e default -p read+,execute+
   ```

   The **mqsichangefileauth** command saves the permissions to the `node.conf.yaml` or `server.conf.yaml` overrides file.
3. Restart the integration node or independent integration server for the changes to take effect.

What to do next

For information about authentication, see “Authenticating users for administration” on page 2507.

Checking file-based permissions
Use the **mqsireportfileauth** command or YAML configuration files to view the file-based administration security permissions that are in effect for the specified integration node.

Before you begin

Read the following topics:

- “Administration security overview” on page 2497
- “Authorizing users for administration” on page 2516
- “Configuring administration security to use file-based, queue-based, or LDAP authorization” on page 2523

About this task

You can display the file-based administration security permissions for the specified integration node by using the **mqsireportfileauth** command or you can view the properties in the `node.conf.yaml` file (for an integration node and its managed integration servers) or the `server.conf.yaml` file (for an independent integration server).

Three levels of authorization are supported for IBM App Connect Enterprise administration security: read, write, and execute. These permissions can be applied to the following types of objects for each role (system user):

- Integration node resources
- Integration server resources
- Data capture objects (record-replay)

Procedure

1. Ensure that the user running the command is a member of the group mqbrkrs.

2. Use the **mqsireportfileauth** command to display the administration security permissions that have been set for a specified role on the integration node.
   For example, display the permissions that are currently in effect for the role `iibAdmins` on the integration node `IB10NODE`:

   ```
   mqsireportfileauth IB10NODE -r iibAdmins
   ```

   The output from the command has a format similar to that shown in the following example:

   ```
   BIP8931I: Role = 'iibAdmins', Resource = '', Permissions = 'read+,write+,execute+'
   ```

3. You can also use the **mqsireportfileauth** command to display all the administration security permissions that have been set for all roles on the integration node. Roles are reported only if they have positive permissions set; any roles that have no permissions set are not reported in the output of this command.
   For example, display all roles for which permissions have been set on the integration node `IB10NODE`:

   ```
   mqsireportfileauth IB10NODE -l
   ```

   The output from the command has a format similar to that shown in the following example:

   ```
   BIP8931I:  Role = 'iibAdmins' Resource = '' Permissions = 'read+,write+,execute+'
   BIP8931I:  Role = 'iibGuests' Resource = '' Permissions = 'read+,write-,execute-'
   ```

   You can also display roles for which permissions have been set on a specified integration server in the integration node; for example:

   ```
   mqsireportfileauth IB10NODE  -e is01  -l
   ```

   The output from the command has a format similar to that shown in the following example:

   ```
   BIP8931I: Role = 'iibAdmins', Resource = 'is01', Permissions = 'read+,write+,execute+'
   BIP8931I: Role = 'iibGuests', Resource = 'is01', Permissions = 'read+,write-,execute-'
   ```

Setting queue-based permissions

You can use IBM MQ queues to authorize users to complete specific tasks against an integration node and its resources.

Before you begin

- Ensure that IBM MQ is installed and that a queue manager is specified on the integration node.
- Read the following topics:
  - “Enabling administration security” on page 2504
  - “Authorizing users for administration” on page 2516

About this task

When you have enabled administration security and specified the queue-based (MQ) authorization mode, you can set the required permissions for users to act on the integration node and its resources. You set the permissions on the following authorization queues:

- SYSTEM.BROKER.AUTH
- SYSTEM.BROKER.AUTH.*EG* (where *EG* is the name of the integration server)
- SYSTEM.BROKER.DC.AUTH

The queue SYSTEM.BROKER.AUTH is created when you use the **mqsichangeauthmode** command to enable queue-based administration security (`mq` mode) on the integration node. When you create an integration server on an integration node for which you have enabled queue-based administration security, the integration server authorization queue SYSTEM.BROKER.AUTH.*EG* is created (if it did not already exist), where *EG* is the name of the integration server.

The SYSTEM.BROKER.DC.AUTH queue is created when you use the **mqsicreatebroker** command to create an integration node with an associated queue manager. For more information about these authorization queues, see "Authorization queues for queue-based administration security" on page 2501.

You can set permissions to individual principals (user IDs), to groups of users, or both, on all platforms:

- If you grant a group or a user ID permissions at the integration node level (on queue SYSTEM.BROKER.AUTH), it does not inherit permissions for integration servers. You must explicitly set permissions for individual integration servers, or for all integration servers.
- On Linux and UNIX, you can authorize both principals and groups. However, when authorizing a principal, IBM App Connect Enterprise additionally authorizes the primary group of that principal. If there are many users who belong to that primary group, they become authorized at the same time. Consider using groups instead of primary groups for authorization, because variants of UNIX use primary groups in different ways.
- If a user ID is a member of the IBM MQ security group mqm, it automatically has permissions to act on all IBM MQ objects.
- On Windows, if a user ID is a member of the security group Administrators, it automatically has permissions to act on all IBM MQ objects.

When you change permissions on a queue, the integration node accesses the updated values the next time that a request is processed. You do not have to stop and restart the integration node.

If you update user ID or group membership by using the operating system facilities on the platform on which the integration node queue manager is running, you must ensure that the queue manager is aware of these changes. Select the option **Refresh Authorization Service** in the IBM MQ Explorer to notify the queue manager of the updated status.

Procedure

1. Ensure that administration security is enabled for the integration node and that the queue-based authorization mode has been set.

   For information about how to enable administration security and set the authorization mode, see "Enabling administration security" on page 2504. For more information about changing the authorization mode, see "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523
2. Follow the steps in one of the following tasks, depending on your platform:
   - "Setting queue-based permissions on Linux, AIX, and Windows systems" on page 2534

Setting queue-based permissions on Linux, AIX, and Windows systems
Use queue-based administration security to grant users permissions to complete specific tasks against an integration node or integration server running on Linux, AIX, or Windows.

Before you begin

Use the command to activate administration security and to specify the queue-based mode of administration security for the integration node or integration server.

About this task

For security reasons, it is important that permissions are set correctly. You can use IBM MQ commands to set up and manage your required security levels. Use the **setmqaut** command to set the required permissions, and the **dspmqaut** command to check which permissions have been set.

The following permissions are required for users to act on the integration node and its resources:

Table 87. Permissions required for acting on an integration node			
Action	**Integration node permission**	**Queue**	**IBM MQ permission (set on setmqaut command)**
View	read	SYSTEM.BROKER.AUTH	+INQ
Create	write	SYSTEM.BROKER.AUTH	+PUT
Delete	write	SYSTEM.BROKER.AUTH	+PUT
Modify	write	SYSTEM.BROKER.AUTH	+PUT
Start	execute	SYSTEM.BROKER.AUTH	+SET
Stop	execute	SYSTEM.BROKER.AUTH	+SET

Table 88. Permissions required for acting on an integration server			
Action	**Integration server permission**	**Queue**	**IBM MQ permission (set on setmqaut command)**
View	read	SYSTEM.BROKER.AUTH.EG	+INQ
Create	write	SYSTEM.BROKER.AUTH.EG	+PUT
Delete	write	SYSTEM.BROKER.AUTH.EG	+PUT
Modify	write	SYSTEM.BROKER.AUTH.EG	+PUT
Start	execute	SYSTEM.BROKER.AUTH.EG	+SET
Stop	execute	SYSTEM.BROKER.AUTH.EG	+SET

Table 89. Permissions required for acting on a record replay and business transaction monitoring object			
Action	**Integration node / integration server permission**	**Queue**	**IBM MQ permission (set on setmqaut command)**
View	read	SYSTEM.BROKER.DC.AUTH	+INQ
	read	SYSTEM.BROKER.DC.AUTH.*integrationServerName*	+INQ
Replay	execute	SYSTEM.BROKER.DC.AUTH.*integrationServerName*	+SET

The **setmqaut** command grants and revokes permissions cumulatively. To avoid retaining unwanted permissions that have been set previously, set them explicitly on each **setmqaut** command by specifying `-all` to remove all existing permissions, followed by the permissions that you want to set.

The following command grants execute permission and retains any permissions that were already set:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 +set
```

The following command grants execute permission and does not retain any existing permissions:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set
```

You can also set multiple permissions at the same time. For example, the following command removes any existing permissions and then grants execute and write permissions:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set +put
```

Use the **dspmqaut** command after each **setmqaut** command, to check that the permissions have been set correctly.

For further information about the commands shown in the following examples, and for details of the parameters, see the IBM MQ product documentation online.

Examples

All the examples shown here are for an integration node that is associated with the queue manager `test`.

Grant only execute permission to the integration node to the user IDs that are defined in the group group1:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1 -all +set
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group1
```

Grant only execute and write permission to the integration node to the user IDs that are defined in the group group2:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2 -all +set +put
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2
```

Revoke execute permission from the user IDs that are defined in the group group2:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2 -set
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH -g group2
```

Using a generic IBM MQ profile on a Linux system, grant only write permission for all integration servers for the user IDs that are defined in the group group3:

```
setmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3 -all +put
dspmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3
```

Note: You enclose generic profile names in quotes on AIX and Linux systems. For more information see the IBM MQ product documentation online and search for the "Using OAM generic profiles on UNIX systems and Windows" topic.

Using a generic IBM MQ revoke write permission on a Linux system for all integration servers for the user IDs that are defined in the group group3:

```
setmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3 -all -put
dspmqaut -m test -t queue -n "SYSTEM.BROKER.AUTH.**" -g group3
```

Grant only read permission for a specific integration server called `default` for group group4:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group4 -all +inq
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group4
```

Revoke execute and write permission for a specific integration server called `default` for group group5:

```
setmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group5 -set -put
dspmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.default -g group5
```

Using a generic IBM MQ on a non-Linux system, dump all IBM MQ permissions for all integration servers:

```
dmpmqaut -m test -t queue -n SYSTEM.BROKER.AUTH.**
```

Configuring authorization by using LDAP groups

Authorize roles in App Connect Enterprise against a Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS) server.

Before you begin

Read the following topics:

- "Administration security overview" on page 2497
- "Authorizing users for administration" on page 2516
- "Configuring administration security to use file-based, queue-based, or LDAP authorization" on page 2523

About this task

You can grant and revoke administration authority for an integration node and for each of its managed integration servers. You can also grant and revoke administration authority for an independent integration server that is not managed by an integration node. You can do these tasks by configuring LDAP authorization for specified groups or attributes in LDAP to specified roles in App Connect Enterprise. You can configure the authorization by setting properties in the `node.conf.yaml` file for the integration node, or the `server.conf.yaml` file for the integration server.

LDAP authorization can be applied only to LDAP authenticated users. LDAP users must belong to one or more LDAP groups, or have one or more LDAP attributes that map to roles in App Connect Enterprise, with appropriate access to the admin REST API. Roles in App Connect Enterprise are granted read, write, or execute permissions for objects in integration nodes or independent integration servers (which are not managed by an integration node). For more information, see "Role-based security" on page 2503. LDAP users can belong to a single LDAP group that can be mapped to a single role in App Connect Enterprise, or multiple LDAP groups that can be mapped to multiple roles in App Connect Enterprise. An LDAP authenticated user's LDAP attributes can also be used to map to roles in App Connect Enterprise.

Configure LDAP authorization by completing the following steps.

Procedure

1. LDAP authorization can be applied only to LDAP authenticated users. If LDAP authentication is not enabled, enable it now as described in "Enabling LDAP authentication" on page 2511.
2. Open the `node.conf.yaml` file for the integration node, or the `server.conf.yaml` file for the integration server by using a YAML editor.

   If you are not able to access a YAML editor, you can edit the file by using a plain text editor. You must ensure that you do not include any tab characters. Tab characters are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   If you run commands that modify settings for an integration node, or an integration server that is managed by an integration node, the modified settings are saved in a `.conf.yaml` file. The `.conf.yaml` file is located in the overrides directory. For example, `C:\ProgramData\IBM\MQSI\components\acev11node\overrides\node.conf.yaml` for an integration node, and `C:\ProgramData\IBM\MQSI\components\acev11node\servers\myNodeServer\overrides\server.conf.yaml` for an integration server managed by an integration node. If a property is set in the integration node's base `node.conf.yaml` file, and also in the overrides directory (`\overrides\node.conf.yaml`), the property value that is set in the overrides directory is used. For more information, see "Configuring an integration node by modifying the node.conf.yaml file" on page 118.

   If you have previously run commands that modify the independent integration server, those modified settings are saved in a `.conf.yaml` file in the overrides directory, for example, `<work directory>/overrides/server.conf.yaml`. If a property is set in the integration server's

`server.conf.yaml` file, and also in the overrides directory (`/overrides/server.conf.yaml`), the property value that is set in the overrides directory is used. For more information, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

3. Enable authorization by setting the **authorizationEnabled** property value to `true` in the `.yaml` configuration file:

```
  #  Authorization
  authorizationEnabled: true     # Clients web user role will be authorized when set true
```

4. Set the **authorizationMode** property to `'ldap'` in the `.yaml` configuration file:

```
authorizationMode: 'ldap'       # Set authorization mode. Choose 1 of : ldap, file or mq
```

5. Optional: Create roles in App Connect Enterprise by configuring the `.yaml` configuration files.

   The roles **viewRole** and **adminRole** are supplied by default in the `.yaml` configuration file. The `node.conf.yaml` and `server.conf.yaml` files allow free-form names for roles that are not predefined in App Connect Enterprise. In the following example, the roles **supportRole**, **manager**, **administrator** and **developer** are created.

```
Permissions:
    # Set Admin Security Authorization file permissions by web user role using 'read+:write
+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission
    # e.g. define the following web user roles 'viewRole' and 'adminRole'
    viewRole:  'read+:write-:execute-'
    adminRole: 'all+'


    supportRole:'read+:write-:execute+'
    manager: 'read+:write-:execute-'
    administrator:'all+'
    developer: 'read+:write-:execute+'
```

6. Set admin security authorization file permissions for roles in the `.yaml` configuration file, by specifying a combination of `read`, `write`, and `execute`, or `all`, with + to grant permission, and - to deny permission, for example:

```
Permissions:
    # Set Admin Security Authorization file permissions by web user role using 'read+:write
+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission
    # e.g. define the following web user roles 'viewRole' and 'adminRole'
    viewRole:  'read+:write-:execute-'
    adminRole: 'all+'
```

   In this example, users with the role `viewRole` are granted read permission only, and users with the role `adminRole` have permission for all actions.

   This step is identical whether you are using LDAP authorization or file-based authorization. For more information, see "Setting file-based permissions" on page 2530.

7. Optional: Set admin security authorization file permissions for roles in the `.yaml` configuration file for each named integration server managed by an integration node. Specify a combination of `read`, `write`, and `execute`, or `all`, with + to grant permission, and - to deny permission, for example:

```
Server:
    # Set Admin Security Authorization file permissions for each named Integration Server
       #server01:
         #Permissions:
            viewRole:  'read+:write-:execute-'
            adminRole: 'all+'
       #server02:
          #Permissions:
            viewRole:  'read+:write-:execute+'
```

```
            adminRole: 'all+'
```

In this example, users with the role `adminRole` have permission for all actions on server01 and server02. Users with the role `viewRole` are granted read permission only on server01, and read and execute permission on server02.

This step is identical whether you are using LDAP authorization or file-based authorization. For more information, see “Setting file-based permissions” on page 2530. By default, integration servers that are managed by an integration node inherit permissions from the integration node.

8. Optional: Map values of LDAP attributes on LDAP objects to names of roles in App Connect Enterprise by configuring the `.yaml` configuration file, as shown in the following example:

```
Security:
  LdapAuthorizeAttributeToRoleMap:
    'graham': 'adminRole'    # (user mapped to role)
    'martin': 'viewRole'    # (user mapped to role)
    'DB_ADMIN':'adminRole'      # (field mapped to role)
    'NETWORK_ADMIN':'viewRole'  # (field mapped to role)
    'o=AceViewersGroup,o=groups,ou=ace': 'viewRole'  # (group mapped to role)
    'o=AceAdminsGroup,o=groups,ou=ace':  'viewRole'  # (group mapped to role)
```

If you have existing LDAP attributes and existing roles in App Connect Enterprise with different names, you must map from one to the other. If LDAP objects have attribute names that do not match names of defined roles in App Connect Enterprise, access is denied. You do not need to map in the following cases:

- If values of LDAP group attributes match the names of roles in App Connect Enterprise.
- If you configure LDAP attribute values to match names of existing roles in App Connect Enterprise.
- If you create roles in App Connect Enterprise with names identical to values of LDAP attributes.

The `node.conf.yaml` and `server.conf.yaml` files allow free-form names for roles that are not predefined in App Connect Enterprise.

LDAP allows free-form names for values of attributes. The names of values of attributes are not predefined in LDAP.

Note:

In LDAP, roles in App Connect Enterprise are derived from LDAP group search attributes. For example, cn, dn, email, location.

Multiple group membership gives the user multiple roles in App Connect Enterprise. For example, AceAdmin, AceViewer, AceDeveloper. When a user tries to perform an action, all roles are tried until permission is granted.

Template syntax allows LDAP to use information about the authenticated user to search for groups. In the **`ldapAuthorizeUrl`** parameters in the `.yaml` configuration file, you can use {{dn}} to substitute the users distinguished name, and {{username}} to substitute the username.

```
#ldapAuthorizeUrl: ldap[s]://server[:port]/baseDN[?[attr_name][?[base|sub]][?filter_expr]]
 # ldap authorization search url
```

ldap:
(Required) An LDAP URL begins with the protocol prefix "ldap." The protocol prefix "ldap" indicates an entry or entries accessible from the LDAP server that is running on the specified hostname at the specified port number.

s:
(Optional) Specifies whether SSL is used. Default is not to use SSL.

server:
(Required) The name or IP address of the LDAP server to contact. You must use the same server that you used for authentication.

port:
(Required) The port to connect to. Default is 389 (non-SSL). For LDAP servers with SSL enabled, the port is typically 636.

baseDN
(Required) The LDAP Distinguished Name. It identifies the base object of the LDAP search.

attr_name:
(Optional) String used to indicate which attributes are returned from the entry. For example, uid, cn, or email address. Default is *uid*.

base|sub:
(Optional) Specifies the scope of the search to run in the specified LDAP server. The allowable scopes are "base" for a base object search, or "sub" for a subtree search. If `base` is defined, the authentication is faster because the DN of the user can be constructed from the *attr_name*, and *baseDN* values. If *sub* is selected, a search must be run before the DN can be resolved. Default is *sub*.

filter_expr:
(Optional) A valid LDAP filter expression that can use template syntax to substitute the details of the authenticated user; for example:

- `{{dn}}` is replaced with the user's distinguished name.
- `{{username}}` is replaced with the user's username attribute value.

Therefore, you can use filters like the following examples.

- `(member={{dn}})` matches groups where the authenticated user's distinguished name is set as a *member* attribute value.
- `(cn={{username}})` matches an entry where the authenticated user's username is set as the *cn* attribute value.

You now have options on how to configure LDAP authorization.

- Authorize a single LDAP group to have a role in App Connect Enterprise. See step “9” on page 2540.
- Authorize multiple LDAP groups to have roles in App Connect Enterprise. See step “10” on page 2541.
- Authorize an LDAP authenticated user's LDAP attributes to have a role in App Connect Enterprise. See step “11” on page 2542.

9. Optional: Authorize a single LDAP group to have a role in IBM App Connect Enterprise by setting values for **`ldapAuthorizeUrl`** in the `.yaml` configuration file.

   Before you do this step, ensure that you set admin security authorization file permissions as required for roles in App Connect Enterprise. Also, ensure that you complete any required mapping of LDAP

attribute names to names of roles in App Connect Enterprise.See step “5” on page 2538, step “6” on page 2538, step “7” on page 2538, and step “8” on page 2539.

In the following example, an LDAP authenticated user, `graham`, is a member of a single group in LDAP, which has the distinguished name (dn) `cn=administrator,ou=groups,o=ace`, as shown in the following LDAP configurations:

```
ObjectClass: Person
dn: cn=graham,ou=users,o=ace
cn: graham


ObjectClass: groupOfNames
dn: cn=administrator,ou=groups,o=ace
cn: administrator
Member: cn=graham,ou=users,o=ace
Member: cn=martin,ou=users,o=ace
```

To authorize the single LDAP group with distinguished name (dn) `cn=administrator,ou=groups,o=ace` to have the role `adminRole` in IBM App Connect Enterprise, set the values for **`ldapAuthorizeUrl`** in the `.yaml` configuration file as shown in the following example:

- `localhost` is the server
- `:10389` is the port
- `o=groups,ou=ace` is the baseDN (Distinguished name)
- `?cn` is the attr_name
- `?sub` indicates that a subset of the LDAP tree structure is to be searched
- `?(member={{dn}})` is the filter_expr

```
ldapAuthorizeUrl: 'ldap://localhost:10389/o=groups,ou=ace?cn?sub?(member={{dn}})'  # ldap
 authorization search url'
```

When the LDAP-authenticated user `graham` attempts to complete an action on the integration node or integration server, a search confirms that the LDAP distinguished name (dn) `cn=administrator,ou=groups,o=ace` is authorized to have the role `adminRole` in App Connect Enterprise. The LDAP-authenticated user `graham` is a member of the LDAP group `cn=administrator,ou=groups,o=ace`, so permission to complete the action is granted.

For more information about the parameters in **`ldapAuthorizeUrl`**, see step “8” on page 2539.

10. Optional: Authorize multiple LDAP groups to roles in App Connect Enterprise by setting values for **`ldapAuthorizeUrl`** in the `.yaml` configuration file.

    Before you do this step, ensure that you set admin security authorization file permissions as required for roles in App Connect Enterprise. Also, ensure that you complete any required mapping of LDAP attribute names to names of roles in App Connect Enterprise. See step “5” on page 2538, step “6” on page 2538, step “7” on page 2538, and step “8” on page 2539.

    In the following example, the LDAP-authenticated user, `martin`, is a member of two groups in LDAP, with the distinguished names (dn) `cn=administrator,ou=groups,o=ace,` and `cn=viewer,ou=groups,o=ace`, as shown in the following LDAP configurations:

```
ObjectClass: Person
dn: cn=martin,ou=users,o=ace
cn: martin

ObjectClass: groupOfNames
dn: cn=administrator,ou=groups,o=ace
cn: administrator
Member: cn=graham,ou=users,o=ace
Member: cn=martin,ou=users,o=ace

ObjectClass: groupOfNames
dn: cn=viewer,ou=groups,o=ace
```

```
cn: viewer
Member: cn=graham,ou=users,o=ace
Member: cn=martin,ou=users,o=ace
```

To authorize the LDAP group with distinguished name (dn) `cn=administrator,ou=groups,o=ace` to have the role `adminRole` in App Connect Enterprise, and authorize the LDAP group with distinguished name (dn) `cn=viewer,ou=groups,o=ace` to have the role, `viewRole` in App Connect Enterprise, configure the .conf.yaml file with the values in the following example:

- `localhost` is the server
- `:10389` is the port
- `o=groups,ou=ace` is the baseDN (Distinguished name)
- `?cn` is the attr_name
- `?sub` indicates that a subset of the LDAP tree structure is to be searched
- `?(member={{dn}})` is the filter_expr

```
ldapAuthorizeUrl: 'ldapAuthorizeUrl: 'ldap://localhost:10389/o=groups,ou=ace?cn?sub?
(member={{dn}})'  # ldap authorization search url'
```

When the LDAP-authenticated user `martin` attempts to complete an action on the integration node or integration server, a search confirms that the LDAP distinguished name (dn) `cn=viewer,ou=groups,o=ace` is authorized to have the role `viewRole` in App Connect Enterprise. The LDAP-authenticated user `martin` is a member of the LDAP group `cn=viewer,ou=groups,o=ace`. Therefore, the user `martin` is granted the permissions that are set for the role `viewRole` in the `.yaml` configuration file of the integration server or integration node. See step “6” on page 2538 and step “7” on page 2538. Each role is checked until permission is either granted or rejected. The LDAP-authenticated user `martin` is also a member of the LDAP group `cn=administrator,ou=groups,o=ace`. Therefore, the user `martin` is granted the permissions that are set for the role `adminRole` in the `.yaml` configuration file of the integration server or integration node.

For more information about the parameters in **`ldapAuthorizeUrl`**, see step “8” on page 2539.

11. Optional: Authorize an authenticated user's LDAP attributes to have a role in App Connect Enterprise by setting values for **`ldapAuthorizeUrl`** in the `.yaml` configuration file.

    Before you do this step, set admin security authorization file permissions as required for roles in App Connect Enterprise. Also, complete any required mapping of LDAP attribute names to names of roles in App Connect Enterprise.See step “5” on page 2538, step “6” on page 2538, step “7” on page 2538, and step “8” on page 2539.

    In the following example, the LDAP attribute `job` has the value `administrator` as shown in the following LDAP configuration:

```
ObjectClass: Person
dn: cn=graham,ou=users,o=ace
cn: graham
```

```
  job:administrator
```

To authorize the LDAP attribute `job` with the value `administrator` to have the role `adminRole` in App Connect Enterprise, configure the `.conf.yaml` file with the values that are shown in the following example:

- `localhost` is the server
- `:10389` is the port
- `o=users,ou=ace` is the baseDN (Distinguished name)
- `?job` is the attr_name
- `?sub` indicates that a subset of the LDAP tree structure is to be searched
- `cn={{username}}` is the filter_expr

```
ldapAuthorizeUrl: 'ldap://localhost:10389/o=users,ou=ace?job?sub?(cn={{username}}) # ldap
 authorization search url'
```

The LDAP-authenticated user `graham` attempts to complete an action on the integration node or integration server. A search confirms that the LDAP attribute name `job`, with the value `administrator` is authorized to have the role `adminRole` in App Connect Enterprise. The LDAP-authenticated user `graham` has an LDAP attribute `job` with the value, `administrator`, so permission to complete the action is granted.

For more information about the parameters in **`ldapAuthorizeUrl`**, see step “8” on page 2539.

12. Save the modified file.
13. Restart the integration node or integration server for the changes to take effect.

Example

The following example shows a `.conf.yaml` file that is configured to enable LDAP authentication and LDAP authorization. It demonstrates the use of free-form names that are allowed by LDAP and App Connect Enterprise. In the section `LdapAuthorizeAttributeToRoleMap`, the LDAP group `developers` is mapped to the role `supportRole` in App Connect Enterprise. The section `Permissions` includes some further examples of roles that are defined in App Connect Enterprise. For example, the role `manager` is defined with read permission.

```
# Admin Security
  #  Authentication
  basicAuth: true
  ldapUrl: 'ldap://localhost:10389/ou=users,o=ace?cn?sub'
  #ldapBindDn: ldap::adminAuthentication
  #ldapBindPassword: ldap::adminAuthentication
  #  Authorization
  authorizationEnabled: true     # Clients web user role will be authorized when set true
  authorizationMode: 'ldap'       # Set authorization mode. Choose 1 of : ldap, file or mq

  ldapAuthorizeUrl: 'ldap://localhost:10389/o=users,ou=ace?description?sub?(member={{dn}})'
  #ldapAuthorizeUrl: 'ldap://localhost:10389/o=groups,ou=ace?employeeType?sub?(cn={{username}})'



Security:
  LdapAuthorizeAttributeToRoleMap:
    'graham': 'adminRole'    # (user mapped to role)
    'martin': 'viewRole'    # (user mapped to role)
    'DB_ADMIN':'adminRole'      # (field mapped to role)
    'NETWORK_ADMIN':'viewRole'  # (field mapped to role)
    'o=AceViewersGroup',o=groups,ou=ace': 'viewRole'  # (group mapped to role)
    'o=AceAdminsGroup',o=groups,ou=ace':  'viewRole'  # (group mapped to role)

    'administrators':'adminRole'
    'developers': 'supportRole'
    'managers': 'viewRole'

  Permissions:
    # Set Admin Security Authorization file permissions by web user role using 'read+:write
+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission
```

```
    # e.g. define the following web user roles 'viewRole' and 'adminRole'
    viewRole:  'read+:write-:execute-'
    adminRole: 'all+'
    supportRole:'read+:write-:execute+'

    #manager: 'read+:write-:execute-'
    #administrator:'all+'
    #developer: 'read+:write-:execute+'

Server:
    # Set Admin Security Authorization file permissions for each named Integration Server
       #server01:
         #Permissions:
            viewRole:  'read+:write-:execute-'
            adminRole: 'all+'
       #server02:
          #Permissions:
            viewRole:  'read+:write-:execute+'
            adminRole: 'all+'
```

Controlling access to data and resources in the web user interface

Integration administrators can control web users' access to data and integration node resources by assigning permissions to users based on their role.

Before you begin

- Read the following topics:
  - web user interface
  - “Role-based security” on page 2503
- Ensure that the web user interface has been configured. For more information, see “Configuring the IBM App Connect Enterprise web user interface” on page 87.

About this task

Integration administrators can restrict web users' access to data and integration node resources only if administration security is enabled. If administration security is not enabled, web users can interact with the web user interface without logging on, which means that they can access the web user interface as the 'default' user and have access to all data and integration node resources.

To perform any administrative task from the web user interface when administration security is enabled, you must have permission to view properties on the integration node. For a full list administrative tasks and the permissions required, see Tasks and authorizations for administration security.

With administration security enabled, REST users can view only the URIs for which they are authorized. If administration security is disabled, all REST requests are unrestricted.

Note: When queue-based security is enabled, a check is made on all SYSTEM.BROKER.AUTH queues to establish the permissions that the user has. As a result of this check, AMQ8077 messages might be seen.

As an integration administrator, you can set permissions to restrict users' access based on the tasks that they are required to perform. Some example tasks and their associated permissions are shown in the following table:

Example access and actions	IBM MQ queue-based permissions (set on the setmqaut command)	File-based permissions (set on the mqsichangefileauth command)
Allow data technicians to view record and replay data stores under **Data** tab of the integration node or integration server in the web user interface.	• `+inq` permission on SYSTEM.BROKER.DC.AUTH queue • `+inq` permission on SYSTEM.BROKER.AUTH queue	• `read+` permission on the `integration node` • `read+` permission on the `Integration Node Data object`

Example access and actions	IBM MQ queue-based permissions (set on the setmqaut command)	File-based permissions (set on the mqsichangefileauth command)
Allow web users to view and download recorded messages in an integration server's record and replay store.	• +inq permission on SYSTEM.BROKER.DC.AUTH queue • +inq permission on SYSTEM.BROKER.DC.AUTH.*integrationServerName* queue • +inq permission on SYSTEM.BROKER.AUTH queue	• read+ permission on the integration node • read+ permission on the Integration Node Data object • read+ permission on the Integration Server Data object
Allow web users to view, download and replay recorded messages in an integration server's record and replay data store.	• +inq permissions on SYSTEM.BROKER.AUTH queue • +inq permission on the SYSTEM.BROKER.DC.AUTH queue • +inq + set permission on SYSTEM.BROKER.DC.AUTH.*integrationServerName* queue	• read+ permission on the integration node • read+ permission on the Integration Node Data object • read+, execute+ permission on the Integration Server Data object
Allow REST users to request information about messages recorded under an integration server's record and replay data store.	• +inq permission on SYSTEM.BROKER.DC.AUTH.*integrationServerName* queue	read+ permission on the Integration Server Data object
Allow REST users to view and replay messages.	• +inq permission on SYSTEM.BROKER.DC.AUTH.*integrationServerName* queue to view • + set permission on SYSTEM.BROKER.DC.AUTH.*integrationServerName* queue to replay	• read+ permission on the Integration Server Data object to view. • execute+ permission on the Integration Server Data object to replay.
Allow web users to view business transactions on the Business Transactions Monitor tab	• +inq permission on SYSTEM.BROKER.AUTH queue • +inq permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server that hosts the business transaction definition • +inq permission on SYSTEM.BROKER.DC.AUTH queue	• read+ permission on the integration node • read+ permission on the Integration Server that hosts the business transaction definition • read+ permission on the integration node data object

Example access and actions	IBM MQ queue-based permissions (set on the setmqaut command)	File-based permissions (set on the mqsichangefileauth command)
Allow web users to view business transaction definitions on the Business Transactions Configure tab	• +inq permission on SYSTEM.BROKER.AUTH queue • +inq permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server that hosts the business transaction definition • +inq permission on SYSTEM.BROKER.DC.AUTH queue	• read+ permission on the integration node • read+ permission on the integration server that hosts the business transaction definition • read+ permission on the integration node data object
Allow web users to start and stop recording for a business transaction definition on the Business Transactions Configure tab.	• +inq permission on SYSTEM.BROKER.AUTH queue • +inq +set permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server that hosts the business transaction definition • +inq permission on SYSTEM.BROKER.DC.AUTH queue	• read+ permission on the integration node • read+ execute+ permission on the integration server that hosts the business transaction definition • read+ permission on the integration node data object
Allow web users to create or update a business transaction definition or a business transaction policy on the Business Transaction Monitoring Configure tab.	• +inq permission on SYSTEM.BROKER.AUTH queue • +inq +put permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server where the business transaction definition is being defined • +inq permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server from which you want to select business monitoring events to include in the business transaction definition • +inq permission on SYSTEM.BROKER.DC.AUTH queue	• read+ permission on the integration node • read+ write+ permission on the integration server where the business transaction definition is defined • read+ permission on the integration server from which you want to select business monitoring events to include in the business transaction definition • read+ permission on the integration node data object

Example access and actions	IBM MQ queue-based permissions (set on the setmqaut command)	File-based permissions (set on the mqsichangefileauth command)
Allow web users to delete a stopped business transaction definition.	• `+inq` permission on SYSTEM.BROKER.AUTH queue • `+inq +put` permission on SYSTEM.BROKER.AUTH.*integrationServerName* queue, where *integrationServerName* is the integration server where the business transaction definition is defined • `+inq` permission on SYSTEM.BROKER.DC.AUTH queue	• `read+` permission on the integration node • `read+ write+` permission on the integration server where the business transaction definition is defined • `read+` permission on the integration node data object

Integration administrators can also allow web users to start and stop integration servers, applications, and message flows from the web user interface, by granting permissions to the roles with which the web users are associated.

For more information about role-based access, see “Role-based security” on page 2503 and “Managing web user accounts” on page 2510.

Procedure

1. Enable administration security and configure the integration node to use either file-based or queue-based authorization.

   For more information, see “Enabling administration security” on page 2504.
2. Define the roles and their associated permissions. You can assign permissions to each role that you have identified; for example, you might decide that your web users can be categorized into two main roles: web administrators and web users. Define a role for each of these groups of users (for example, `iibUsers` and `iibAdmins`) with permissions that allow them to perform the required tasks, such as viewing or modifying resources:

   - If the integration node is configured to use file-based authorization (**`file`** mode), you define the roles and associated permissions on the integration node, by using the **mqsichangefileauth** command. For information about setting permissions for file-based authorization, see “Setting file-based permissions” on page 2530.
   - If the integration node is configured to use queue-based authorization (**`mq`** mode), you must create a system user ID on the operating system for each role that you have identified. You must then assign permissions to the system user ID, which is then used as a role. For information about setting permissions for queue-based authorization, see “Setting queue-based permissions” on page 2533.
   - If the integration node is configured to use LDAP authorization, you define the roles and associated permissions on the integration node, by setting properties in the `node.conf.yaml` for the integration node. For information about setting permissions for LDAP authorization, see “Configuring authorization by using LDAP groups” on page 2537.

   For more information about setting the appropriate permissions, see “Authorizing users for administration” on page 2516.
3. Use the **mqsiwebuseradmin** command to create your web user accounts and assign them to the appropriate roles.

   For more information, see command. For more information about roles, see “Role-based security” on page 2503.

Configuring HTTP basic authentication for an integration node or server

You configure HTTP basic authentication for your IBM App Connect Enterprise integration node or server by modifying properties in a `node.conf.yaml` or `server.conf.yaml` configuration file. .

Before you begin

- Start an instance of the IBM App Connect Enterprise command console. You can use the console to create a username and password by issuing the **mqsiwebuseradmin** command.

Procedure

1. Use a YAML editor to open the `.yaml` configuration file for your integration node or server.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your configuration to fail. If choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Uncomment the following line in the `.yaml` file:

   ```
   #basicAuth: true       # Clients require a web username and password
   ```
3. Save the `.yaml` file.

   The properties that you set in the `.yaml` file take effect when the integration node or server is started. If you modify these properties again, you must also restart the integration node or server.
4. In the IBM App Connect Enterprise command console, create a username and password by issuing the **mqsiwebuseradmin** command.
   For example:

   ```
   mqsiwebuseradmin -w c:\workdir\ACEServ1 -u admin -a password -c
   ```

   You can create multiple users; the permissions are identical for all users.
5. Restart the integration node or server for the changes to take effect.

What to do next

When you manage your integration node or server, or the resources that are deployed to it, by using either the IBM App Connect Enterprise Toolkit or web user interface, you are prompted to enter the username and password that you have created in this task in addition to the host name and port number of the integration node or server. When you use the IBM App Connect Enterprise Toolkit to create a connection to the integration node or server, you can optionally save the password to the Eclipse Secure Storage by selecting the **Save Password** check box in the Create connection wizard.

Configuring SSL or TLS for an integration node or server

You configure SSL or TLS for your IBM App Connect Enterprise integration node or server by modifying properties in a `.yaml` configuration file. You also use the **mqsisetdbparms** command to set a password.

Before you begin

- Start an instance of the IBM App Connect Enterprise command console. You can use the console to create a username and password by issuing the **mqsisetdbparms** command.

Procedure

1. Use a YAML editor to open the `.yaml` configuration file for your integration node or server.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your configuration to fail. If choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Uncomment the following lines in the `.yaml` file:

   ```
   #sslCertificate: '/path/to/serverPKCS.p12'  ...
   #sslPassword: 'adminRestApi::sslpwd'        ...
   ```

   Where `adminRestApi::sslpwd` is the default resource name to be specified on the **mqsisetdbparms** command.

   You can specify values inside or without single quotes[2]. Also note the comments in the `.yaml` file about values to specify for the type of server certificate that you want to use.
3. In the line that starts `SslCertificate`, specify the file path to the server certificate on your system. For example, to use a p12 certificate:

   ```
   sslCertificate: '/Work/ACEv11/certificates/ssl/key.p12'   # See comment below
   ```
4. To use a pem certificate, in the line that starts `SslPassword` specify the file path to the pem file. For example:

   ```
   sslPassword: '/Work/ACEv11/certificates/ssl/cakey.pem'   # See comment below
   ```
5. Save the `.yaml` file.

   The properties that you set in the `.yaml` file take effect when the integration node or server is started. If you modify these properties again, you must also restart the integration node or server.
6. To use a p12/pfx certificate, run the **mqsisetdbparms** command to specify the password for your server certificate.

   - Ensure that you specify the resource name on the **-n** parameter as `adminRestApi::sslpwd` to match the `sslPassword` value in the `.yaml` file.
   - The -u (username) value is ignored.

   For example:

   ```
   mqsisetdbparms -w c:\workdir\ACEServ1 -n adminRestApi::sslpwd -u dummy -p password
   ```
7. Restart the integration node or server for the changes to take effect.

What to do next

When you use the IBM App Connect Enterprise Toolkit to create a connection to the integration node or server, ensure that you select the **Use HTTPS** check box in the Create connection wizard.

Disabling administration security

Disable administration security to remove control over an integration node and its resources.

About this task

You can disable administration security for an integration node by updating the **authorizationEnabled** property in the `node.conf.yaml` configuration file.

[2] To be more explicit about the string values, you should put the values inside quotes.

Procedure

1. Open the `node.conf.yaml` configuration file for your integration node, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and will cause your integration node configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. In the `node.conf.yaml` file, set the **`authorizationEnabled`** property to `false`:

   For example:

   ```
   authorizationEnabled: false
   ```
3. Save the modified `node.conf.yaml` file.
4. Restart the integration node by using the web user interface or the command.

Results

When this command completes, all users are able to complete tasks against the integration node. They can also complete all tasks against all integration servers that have been defined, and that are defined in the future, on this integration node.

Message flow security

Control access to individual messages in a message flow, using the identity of the messages.

The following topics introduce the concepts that you need to understand before you can configure message flow security, and they explain the steps involved in setting up security for your message flows:

- “Message flow security overview” on page 2550
- “Setting up message flow security” on page 2583

Message flow security overview

IBM App Connect Enterprise provides a security manager to control access to individual messages in a message flow, by using the identity of the message.

You can configure an integration server for processing end-to-end an identity that is carried in a message through a message flow by using a security profile. By creating a security profile, you can configure security for a message flow to control access based on the identity that is associated with the message and provide a security mechanism that is independent of both the transport type and message format.

Only a subset of the connectors available in IBM App Connect Enterprise use security profiles to control and vary the identity that is used when the connector interacts with an external system. For other connectors, a fixed identity can be specified, which is used to authorize access to the external system. For those connectors, the integration server has its own repository of identities, which can be updated by using the **`mqsivault`** and **`mqsicredentials`** commands.

If you do not enable message flow security, the default security facilities that are in IBM App Connect Enterprise are based on the security facilities that are provided by the transport mechanism. In this case, the integration server processes all messages that are delivered to it, using the integration server service identity as a proxy identity for all message instances. Any identity that is present in the incoming message is ignored.

The security functions that are associated with a message flow are controlled by using “Security profiles” on page 2552, which are created by the App Connect Enterprise administrator and accessed by the security manager at runtime.

The security manager enables the integration server to:

- Extract the identity from an inbound message

- Authenticate the identity, by using either an external security provider (such as LDAP) or a locally stored identity in the integration server vault
- Map the identity to an alternative identity, by using an external security provider
- Check that either the alternative identity or the original identity is authorized to access the message flow, by using an external security provider
- Propagate either the alternative identity or the original identity with an outbound message.

The following external security providers (also known as Policy Decision Points or PDPs) are supported:

- Tivoli Federated Identity Manager (TFIM) V6.1 for authentication, mapping, and authorization
- WS-Trust V1.3 compliant Security Token Service (including TFIM V6.2) for authentication, mapping, and authorization
- Lightweight Directory Access Protocol (LDAP) for authentication and authorization
- Windows Domain Controllers for authentication
- Kerberos KDCs for authentication

As an alternative to using an external security provider to authenticate an identity, you can use an identity stored locally in the integration server vault. For more information, see "Authenticating incoming requests by using credentials stored in the vault" on page 2597.

You can invoke message flow security by configuring either a security enabled input node or a SecurityPEP node. You can use the SecurityPEP node to invoke the message flow security manager at any point in the message flow between an input node and an output (or request) node.

For an overview of the sequence of events that occur when the message flow security manager is invoked by using either a security enabled input node or a SecurityPEP node, see the following topics:

- "Invoking message flow security using a security enabled input node" on page 2565
- "Invoking message flow security using a SecurityPEP node" on page 2569

The input nodes that support message flow security are:

- MQInput
- HTTPInput
- SOAPInput

The support for treating security exceptions as normal exceptions is provided by only the MQInput and HTTPInput nodes; it is not available in the SOAPInput node.

The output nodes that support identity propagation are:

- CICSRequest
- HTTPRequest
- HTTPAsyncRequest
- IMSRequest
- MQOutput
- MQReply
- RESTRequest
- RESTAsyncRequest
- SAPRequest
- SiebelRequest
- SOAPRequest
- SOAPAsyncRequest
- SOAPReply
- SecurityPEP

If the message flow is a web service that is implemented by using the "SOAP nodes" on page 811, the identity can be taken from the "WS-Security" on page 2650 header tokens that are defined through appropriate "Policy sets" on page 2662 and bindings.

To improve performance, the security manager uses a security cache. Entries are created in the security cache when a message flow with a security profile performs authentication, mapping, or authorization. The entries are valid for the length of time that is specified by the `cacheTimeout` property of the `securitycache` component after which the entries are marked as expired. When an entry is marked as expired, it must be reauthenticated, mapped, or reauthorized with the security provider before it can be reused, and its expiry time is reset. For information about specifying a value for the `cacheTimeout` property, see "Configuring the security cache" on page 2579. You can use the **mqsireloadsecurity** command to force the immediate expiry of some or all of the entries in the security cache.

For a SOAPRequest and SOAPAsyncRequest node, an appropriate policy set and bindings can be defined to specify how the token is placed in the WS-Security header (rather than the underlying transport headers). For more information, see "Policy sets" on page 2662.

The following topics in this section provide more detailed information about message flow security:

- "Identity" on page 2554
- "Security profiles" on page 2552
- "Authentication and validation" on page 2561
- "Authorization" on page 2562
- "Identity mapping" on page 2564
- "Identity and security token propagation" on page 2580
- "Security identities for the integration server connecting to external systems" on page 2582
- "Invoking message flow security using a security enabled input node" on page 2565
- "Invoking message flow security using a SecurityPEP node" on page 2569
- "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574
- "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 2572
- "Configuring the security cache" on page 2579
- "Security exception processing" on page 2582

Security profiles

A security profile defines the security operations that are to be performed in a message flow at SecurityPEP nodes and security enabled input and output nodes.

Security profiles are configured by the integration administrator before deploying a message flow, and are accessed by the security manager at run time.

A security profile allows an integration administrator to specify whether authentication, authorization, mapping, and propagation are to be performed on the identity or security tokens associated with messages in the message flow. You can create a security profile for use with an external security provider (also known as a Policy Decision Point or PDP) to provide security enforcement and mapping. IBM Tivoli Federated Identity Manager (TFIM) V6.1 and WS-Trust v1.3 compliant Security Token Service (including TFIM V6.2) are supported for authentication, authorization, and mapping. Lightweight Directory Access Protocol (LDAP) is supported for authentication and authorization. As an alternative to using an external security provider for authentication, you can create a security profile for authenticating against credentials that are held locally in the integration server's vault, as described in "Authenticating incoming requests by using credentials stored in the vault" on page 2597.

Security profiles apply to the following security-enabled message flow nodes:

- CICSRequest
- HTTPInput
- HTTPRequest

- HTTPAsyncRequest
- IMSRequest
- MQInput
- MQOutput
- MQReply
- RESTRequest
- RESTAsyncRequest
- SAPRequest
- SecurityPEP
- SiebelRequest
- SOAPInput
- SOAPReply
- SOAPRequest
- SOAPAsyncRequest

These nodes have a **Security Profile** property, in which you can specify the name of the security profile that determines the type of security that is configured for the node. If the named security profile does not exist at run time, the message flow fails to deploy. If a specified external security provider does not support the type of token configured on the node for the security operation, an error is reported and the message flow fails to deploy. If the security profile specifies local authentication, but the credentials have not been created or the alias name is not matching, the deployed message flow will fail to start. For more information, see "Authenticating incoming requests by using credentials stored in the vault" on page 2597.

Security profiles can be configured by the administrator at deployment time in the BAR file editor. The security-enabled nodes have a **Security Profile** property in the BARfile editor, which can be left blank, set to *No Security*, or set to a specific security profile name. Set *No Security* to explicitly turn off security for the message flow node. If the **Security Profile** property is blank, the node inherits the **Security Profile** property that is set at the message flow level. If the **Security Profile** property is left blank at both levels, security is turned off for the message flow node.

The security profile also specifies whether propagation is required. A pre-configured profile that specifies propagation is provided for use by output and request nodes. This profile is the *Default Propagation* security profile. This profile can also be used on an input node to extract tokens and put them into the message tree ready for propagation or processing in a SecurityPEP node.

Security profiles contain values for the following properties:

alternateServers

Defines the comma-separated list of LDAP servers to failover when the primary server is not available. The list has the following format:

```
ldap[s]://host1:[port1], ldap[s]://host2:[port2], ldap[s]://host3:[port3]
```

After failover, the newly connected LDAP server becomes the primary server.

authentication

Defines the type of authentication that is performed on the source identity. For more information, see "Authentication and validation" on page 2561.

authenticationConfig

Defines the information that the integration server needs to authenticate the identity, and can be one of the following values:

- The configured alias name of the locally stored credentials in the integration server's vault. These credentials are used for basic authentication when the **Authentication** type is `Local`. For more

information, see “Authenticating incoming requests by using credentials stored in the vault” on page 2597.

- The information that the integration server needs to connect to the external security provider and look up identity tokens. This property is in the form of a provider-specific configuration string.

mapping
Defines the type of mapping that is performed on the source identity. For more information, see “Identity mapping” on page 2564.

mappingConfig
Defines how the integration node connects to the provider, and contains additional information required to look up the mapping routine. It is a provider-specific configuration string.

authorization
Defines the types of authorization checks that are performed on the mapped or source identity. For more information, see “Authorization” on page 2562.

authorizationConfig
Defines how the integration node connects to the provider, and contains additional information that can be used to check access (for example, a group that can be checked for membership). It is a provider-specific configuration string.

passwordValue
Defines how passwords are treated when they enter a message flow. If *PLAIN* is selected, the password appears in the Properties folder in plain text. If *OBFUSCATE* is selected, the password appears in the Properties folder in base64 encoding. If *MASK* is selected, the password appears in the Properties folder as four asterisks (****).

propagation
Enables or disables identity propagation on output and request nodes. On the security enabled input nodes, you can choose to select only identity propagation, without specifying any other security operations, to make the extracted incoming identity or security token available for use in the other nodes in the message flow, such as output or request nodes. For more information, see “Identity and security token propagation” on page 2580.

idToPropagateToTransport
Enables the use of a specific security identity for propagation. Set the value to `STATIC ID`, and set the security identity by using the **`transportPropagationConfig`** parameter.

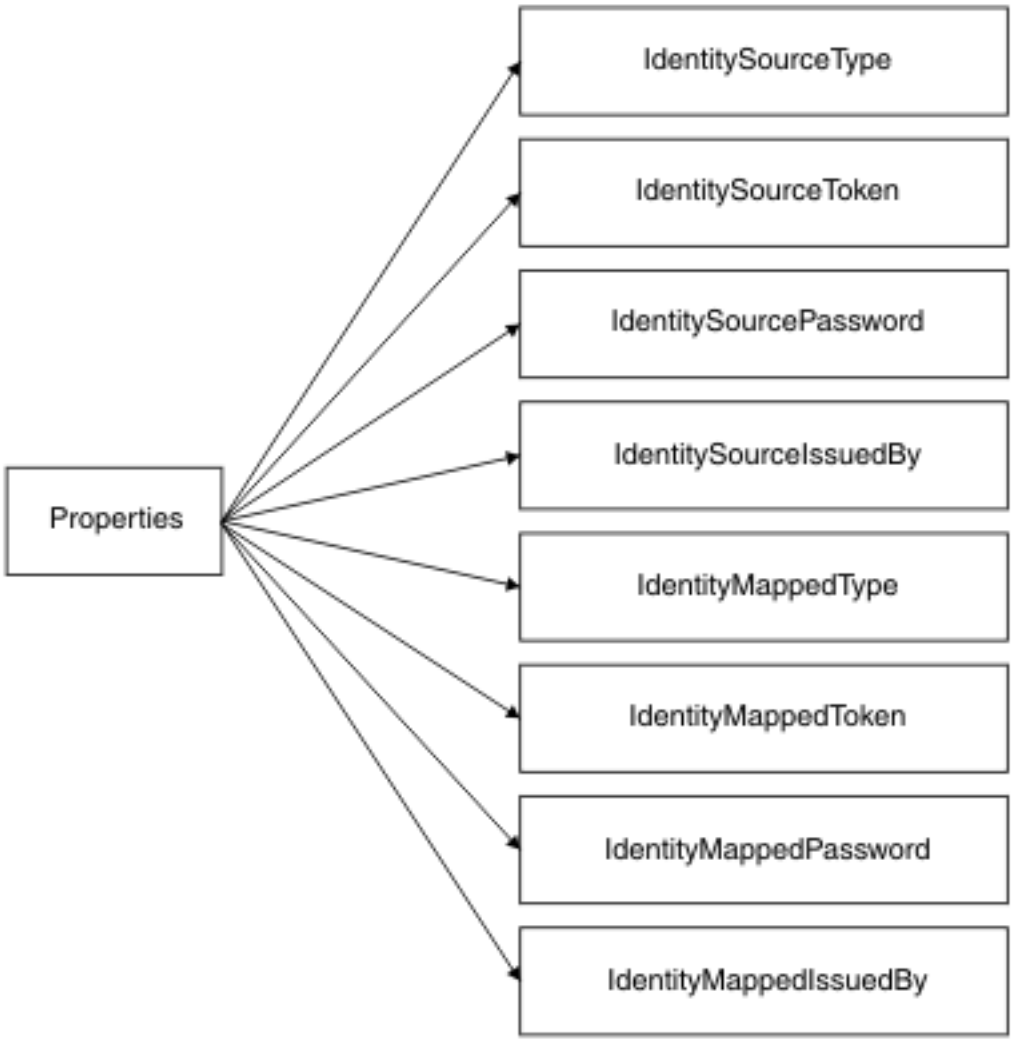
transportPropagationConfig
Provides a specific security identity to propagate when **`idToPropagateToTransport`** is set to `STATIC ID`. Set the value to the name that you associate with the static user name and password identity when you run the **`mqsisetdbparms`**. For more information, see “Configuring a message flow for identity propagation” on page 2613.

For information about configuring settings for the security profile, see “Creating a security profile” on page 2584.

Identity

In IBM App Connect Enterprise, an identity is a security token that uniquely identifies an individual, or that provides a set of assertions that can be validated.

When a SecurityPEP node or a supported input node is configured with a security profile, the extracted identity is held in the integration node as eight properties in the Properties folder of the message tree structure. These properties define two identities in the integration node: source and mapped. For both the source and mapped identities, values are held for **Type**, **Token**, **`Password`**, and **`IssuedBy`** properties:



The **Identity token type** property on the security-enabled input nodes can be set to a value of *Transport Default*, which causes the token type to be created from the default identity header or fields of the transport. For WebSphere MQ, *Transport Default* provides an identity type of *Username*. For HTTP, *Transport Default* provides an identity type of *Username and Password*. The Identity token type property on the SecurityPEP node can be set to *Current Token*, which enables it to use the identity in the Properties folder fields instead of extracting a new identity from the message.

The following table shows the support that is provided (by the message flow security manager and external security providers) for the extraction of different types of security token. For information about the token types that are supported for identity propagation, see "Identity and security token propagation" on page 2580.

Table 90. Support for security token types - token extraction

Token type (format)	Integration node security manager support for token extraction	External security provider support
Username	Username tokens are supported for extraction by the following message flow nodes: • HTTPInput • MQInput • SecurityPEP • SOAPInput The token is obtained from one of the following transport headers: • MQ – From MQMD user ID • HTTP – From HTTP BasicAuth header containing only a username part • SOAP – From a WS-Security UsernameToken header. The policy set and binding (associated with the SOAP node) must define a username profile, and the wsse:UsernameToken must contain only a `wsse:Username` element. – From the Kerberos subject in a WS-Security header. The policy set and binding (associated with the SOAP node) must define a Kerberos profile. – From the HTTP BasicAuth header containing only a username part if no policy set is defined on the SOAP node. Alternatively, the token can be taken from any part of the message tree when the token location is specified (on the node) using an XPath expression or ESQL field path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *username*.	LDAP: Authorization WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization **Note:** To prevent the use of a username token (rather than a username and password token) with these security providers, consider setting the **`Reject Empty Password`** security profile property to TRUE. The security manager then rejects a user name during authentication if the user name has an empty password token, and the user name is not authenticated with the external security provider.

Table 90. Support for security token types - token extraction (continued)

Token type (format)	Integration node security manager support for token extraction	External security provider support
Username and password or Username and RACF PassTicket	Username and password tokens are supported for extraction by the following message flow nodes: • HTTPInput • MQInput • SecurityPEP • SOAPInput The token is obtained from one of the following transport headers: • HTTP – From HTTP BasicAuth header containing both a username and password part • SOAP – From a WS-Security UsernameToken header. The policy set and binding (associated with the SOAP node) must define a username profile and the wsse:UsernameToken must contain both `wsse:Username` and `wsse:Password` elements. – From the HTTP BasicAuth header containing only a username part if no policy set is defined on the SOAP node Alternatively, the token can be obtained from any part of the message tree when the token location is specified (on the node) using an XPath expression or ESQL path. The password token can carry either a clear text password or a RACF PassTicket. If you are using a WS-Trust V1.3 STS (such as TFIM V6.2), you can use it to map (issue) or validate RACF PassTickets, by specifying the token type as *Username and password*. This support is available with security enabled input nodes and SecurityPEP nodes. The literal string value used by the integration node (and which you can use to specify the token	LDAP: • Authentication • Authorization TFIM V6.1: • Authentication • Mapping • Authorization WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization IWA: • Authentication

Table 90. Support for security token types - token extraction (continued)

Token type (format)	**Integration node security manager support for token extraction**	**External security provider support**
SAML assertion	SAML tokens are supported for extraction by the following message flow nodes: • SecurityPEP • MQInput • HTTPInput • SOAPInput The token is obtained from one of the following places: • SOAP – From a WS-Security header. The policy set and binding (associated with the SOAP node) must define a SAML profile. • Any part of the message tree when the token location is specified using an XPath expression or ESQL path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *SAML*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization

Table 90. Support for security token types - token extraction (continued)

Token type (format)	Integration node security manager support for token extraction	External security provider support
Kerberos GSS v5 AP_REQ	Kerberos tickets are supported for processing by the message flow security manager from the SecurityPEP node. A WS-Security Kerberos token profile is supported by the following SOAP nodes, but in this case, the Kerberos Key Distribution Center is communicated with directly, and the properties folder is populated with a Username token representing the Kerberos subject: • SOAPInput The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *kerberosTicket*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization
LTPA v2 token	LTPA tokens are supported for extraction by the following nodes: • SecurityPEP • SOAPInput The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *LTPA*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization

Table 90. Support for security token types - token extraction (continued)

Token type (format)	Integration node security manager support for token extraction	External security provider support
X.509 Certificate	X.509 tokens are supported for extraction by the following message flow nodes: • SecurityPEP • MQInput • HTTPInput • SOAPInput The token is obtained from one of the following places: • SOAP – From a WS-Security header. The policy set and binding (associated with the SOAP node) must define an X.509 profile. • Any part of the message tree when the token location is specified using an XPath expression or ESQL path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *X.509*.	TFIM V6.1: • Authentication • Mapping • Authorization. WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization.
Universal WSSE token	Universal WSSE tokens are supported for extraction by the SecurityPEP node only. The token is obtained from any part of the message tree at a SecurityPEP node, if the token location is specified using an XPath expression or ESQL path. The literal string value used by the integration node (and which you can use to specify the token type in an ESQL or Java program) is *UniversalWsse*.	WS-Trust V1.3 STS (including TFIM V6.2): • Authentication • Mapping • Authorization.

The source identity is set by the SecurityPEP or input node only if a security profile is associated with the node. The information to complete these fields is typically found in the headers of a message but can also be located in the body (provided that the node has been configured with an ESQL Path or XPath reference for the various properties). If multiple identities are available (for example, if you are using message aggregation), the first identity is used. The token extraction is transport specific and can be performed only using transports that support the flow of identities. These transports are: Websphere MQ, HTTP(S), and SOAP. See node and node for more information.

You can modify the values in the properties (for example, from ESQL), but do not write to the *IdentitySource** values. For example, you can create a custom identity mapping routine in ESQL or Java by using the *IdentitySource** values to create custom *IdentityMapped** values.

SAML and Universal WSSE tokens are stored in the Properties tree IdentitySourceToken or IdentityMappedToken field as a character bit stream. To access this data as a message tree, parse it into a suitable parser, such as XMLNSC:

```
-- Parse the mapped SAML2.0 token in the properties folder and set it in the message body
   CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNSC')
 PARSE(InputRoot.Properties.IdentityMappedToken,
   InputProperties.Encoding, InputProperties.CodedCharSetId);
```

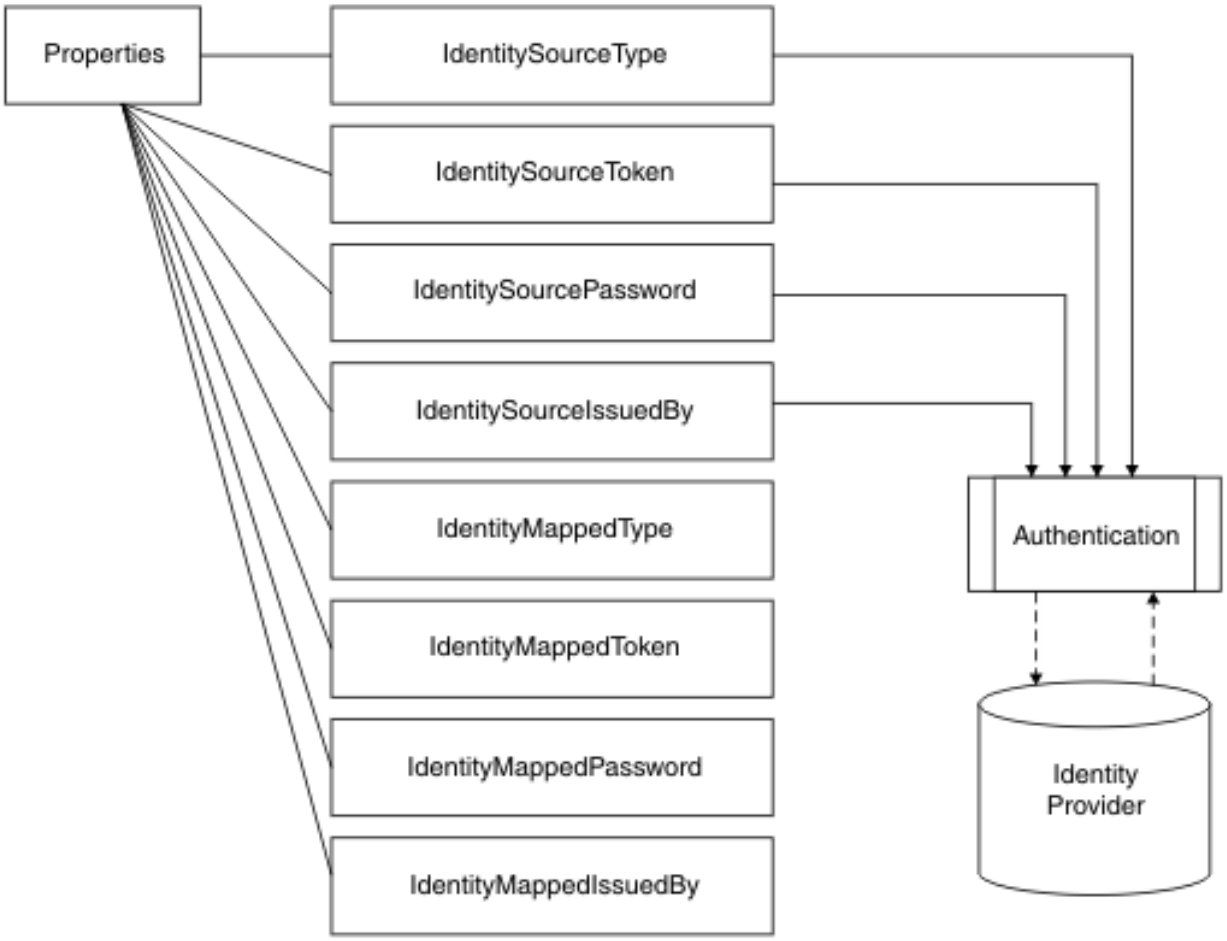
To set either SAML or Universal WSSE tokens into the properties fields, you must obtain the bit stream of a tree; for example, by using the ESQL ASBITSTREAM function.

Authentication and validation

Authentication is the process of establishing the identity of a user or system and verifying that the identity is valid. Applying authentication to a SAML security token involves validating the assertions that it carries and confirming that it is being processed within its validity period.

In IBM App Connect Enterprise message flow security, authentication involves the security manager either passing the identity type and token to an external security provider or checking the identity by using credentials stored locally in the integration server's vault. For more information about security tokens, see “Identity” on page 2554.

Authentication by an external security provider:



The following external security providers (also known as Policy Decision Points) are supported for authentication:

- Lightweight Directory Access Protocol (LDAP)
- Tivoli Federated Identity Manager (TFIM) V6.1
- WS-Trust V1.3 Security Token Service (STS), including TFIM V6.2
- Windows domain controller and Kerberos Key Distribution Center

The external security provider checks the identities and returns a value to confirm whether the identity is authentic. If the identity is not authentic, a security exception is raised.

Consider setting the **`Reject Empty Password`** property to TRUE to specify that you want the security manager to reject a user name during authentication if the user name has an empty password token, without authenticating the user name with the configured provider.

Some identity providers support only a single type of authentication token. If a token of another type is passed into the message flow, an exception is raised. For example, LDAP supports only a *Username and password* token.

You can use an LDAP provider for the authentication of an incoming identity token. The LDAP server must be LDAP Version 3 compliant.

Alternatively, you can use a WS-Trust v1.3 STS provider (for example, TFIM Version 6.2) for the authentication of an incoming identity or security token. The security manager invokes the WS-Trust v1.3 provider once, even if it is set for additional security operations (such as mapping or authorization). As a result, when you are using TFIM, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

For more information about using TFIM V6.2 for authentication, see “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 2574.

TFIM V6.1 is also supported, for compatibility with previous versions of IBM App Connect Enterprise. For more information about using TFIM V6.1 for authentication, see “Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 2572.

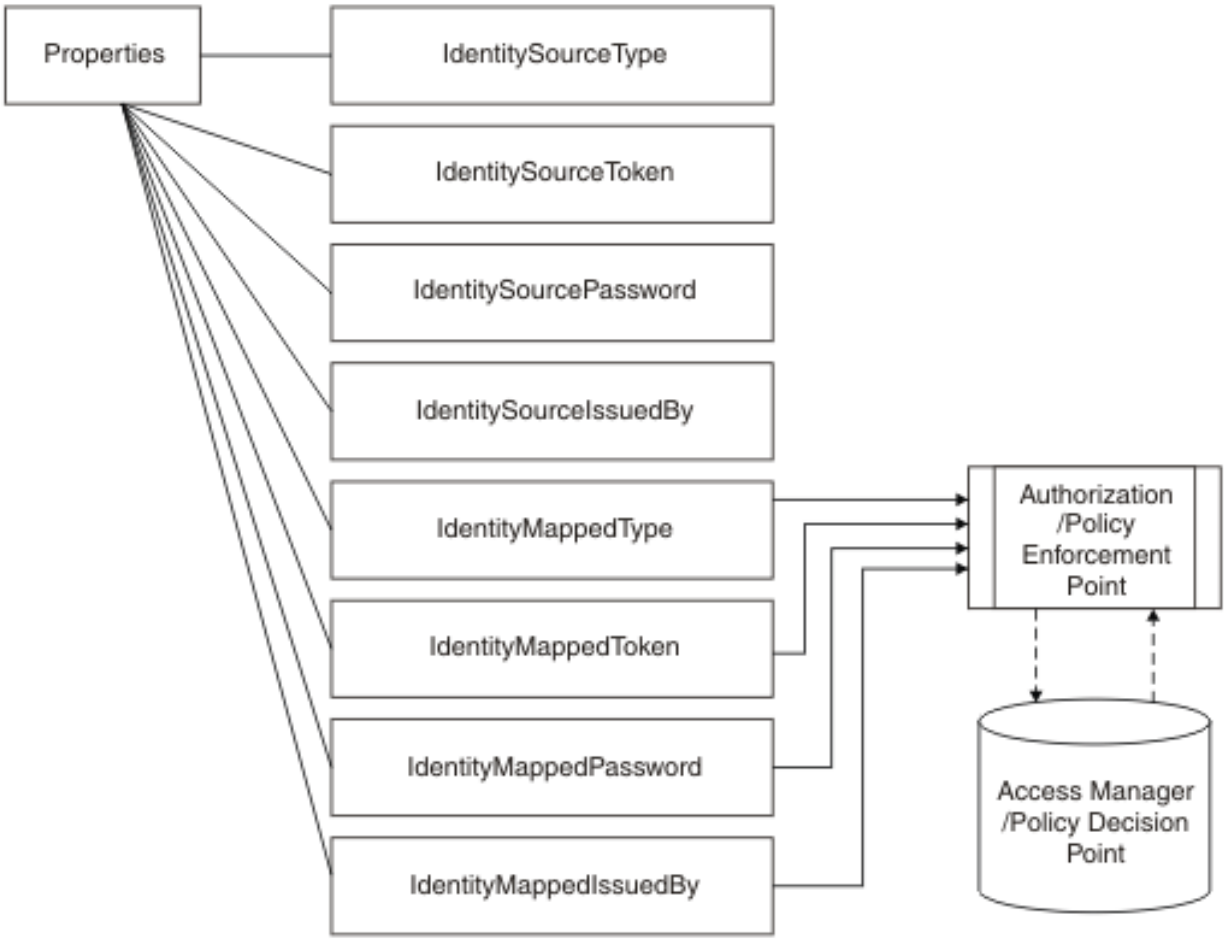
Authentication using locally stored credentials:

As an alternative to using an external security provider, you can implement basic authentication on incoming requests by using credentials stored locally in an independent integration server's vault. For information about using locally stored credentials, see “Authenticating incoming requests by using credentials stored in the vault” on page 2597.

Authorization

Authorization is the process of verifying that an identity token has permission to access a message flow.

If authentication and mapping are configured, they are used to verify the identity before it is authorized.



If a mapped identity exists, authorization is applied to the mapped identity. If a mapped identity does not exist, the source identity is used.

If you specify LDAP as the provider for authorization, the security manager queries the configured LDAP server (which must be LDAP Version 3 compliant), to validate that the identity is a member of the LDAP group that is configured in the security profile.

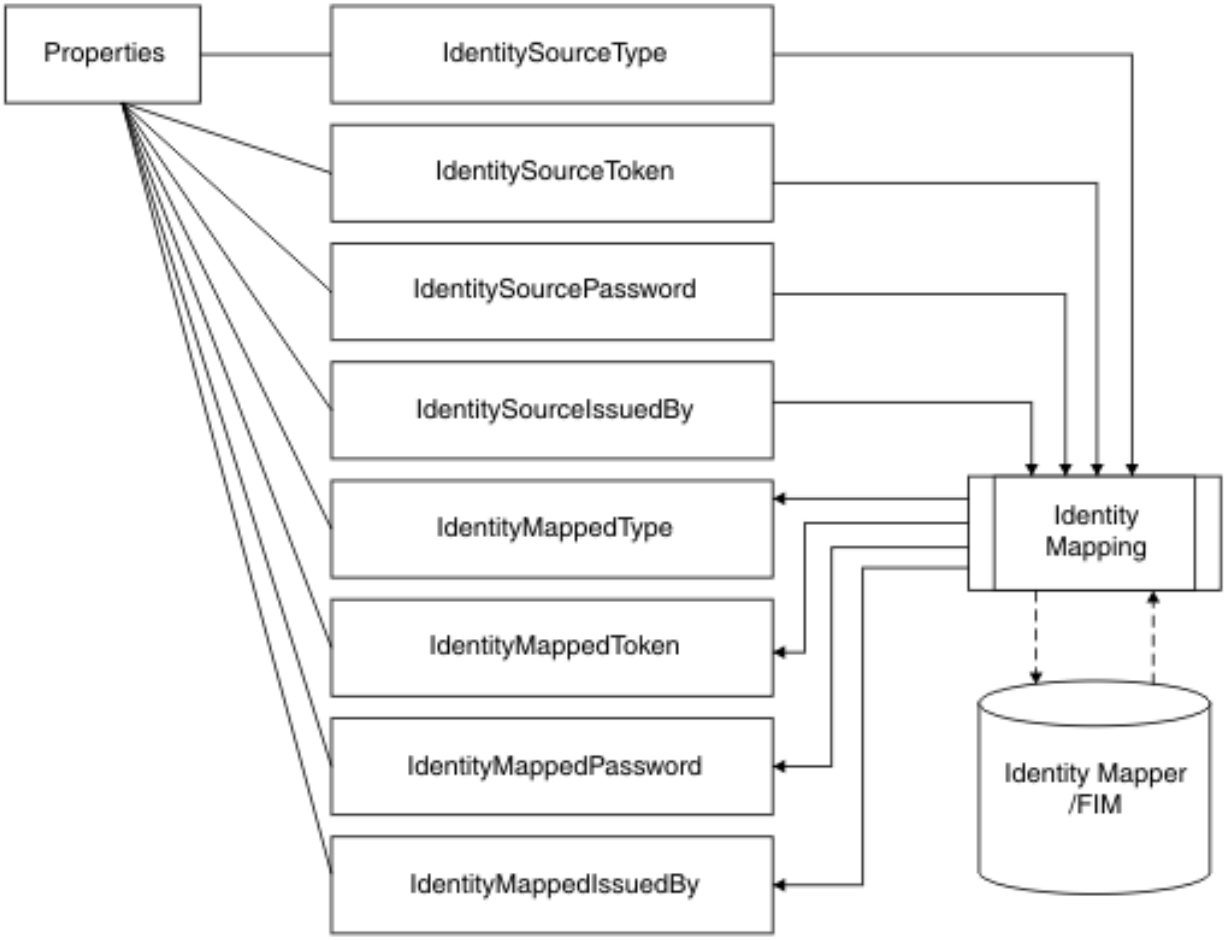
If you specify *WS-Trust v1.3 STS* as the provider for authorization, the security manager invokes the Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to validate that the identity token provided has permission to access the message flow. If you are using TFIM V6.1 rather than TFIM V6.2, you can specify *TFIM* as the provider for authorization.

For more information about using TFIM V6.2 for authorization, see “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 2574.

For information about using TFIM V6.1 for authorization, see “Authentication, mapping, and authorization with TFIM V6.1 and TAM” on page 2572.

Identity mapping

Identity mapping is the transformation of a security token from one format to another format, or the federation of an identity from one realm to an equivalent identity in another realm.



IBM App Connect Enterprise provides support for identity mapping (also known as identity federation) and token issuance and exchange. Identity mapping is the process of mapping an identity in one realm to another identity in a different realm. For example, you might map *User001* from the eSellers realm to *eSellerUser01* in the eShipping realm. Token issuance and exchange involves the mapping of a token of one type to a token of a different type. For example, an incoming Username and Password token from a client over MQ might be mapped into an equivalent SAML assertion, to be propagated to a web services call. Alternatively, you might exchange a SAML 1.1 assertion from a client application for an equivalent SAML 2.0 assertion for an updated backend server.

Mapping using a WS-Trust provider

The IBM App Connect Enterprise security manager supports mapping operations through a WS-Trust V1.3 compliant Security Token Service (STS), such as IBM Tivoli Federated Identity Manager (TFIM) V6.2. Mapping is performed for SecurityPEP nodes and input nodes that have an associated security profile that includes a mapping operation configured with a WS-Trust V1.3 STS.

WS-Trust V1.3 (TFIM V6.2) can also be selected for authentication and authorization in the profile. However, if more than one security operation is associated with the security profile, only one WS-Trust request is issued to the STS. As a result, the STS must be configured to perform all the required operations. For example, if a TFIM V 6.2 server is specified, the TFIM module chain that is invoked must include the appropriate validate, authorize, and issue modules. If you require each operation to be performed through a separate WS-Trust call, you must use a series of SecurityPEP nodes, each associated

with a different security profile that is configured for only one security operation (authentication, authorization, or mapping).

If the security profile specifies only mapping with WS-Trust v1.3 STS, the request is sent with a request type of *Issue*, whereas a mixed set of security operations sends a request type of *Validate*. When mapping is included, the STS must return a token in its response, even if it is the original token; otherwise, an error occurs.

To provide compatibility with previous versions of IBM App Connect Enterprise, support is also provided for TFIM V6.1.

In the integration node, identity mapping is performed at the input node or SecurityPEP node, after authentication but before authorization. The source identity is passed to an identity mapper for processing. If both mapping and authorization are configured, the authorization operation uses the mapped output token rather than the source token, which means that the authorization is performed on the federated identity.

Mapping is not performed in output nodes, even if the output node has been configured with a security profile.

IBM App Connect Enterprise supports mapping between any type of security token that is supported by the configured security provider. For more information about the support provided, see "Identity" on page 2554.

When mapping from an X.509 certificate, TFIM can validate the certificate but cannot verify the identity of the original sender. However, if it is required, this verification integrity check can be performed by the SOAPInput node.

For more information about using TFIM V6.2 for mapping, see "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574.

For information about using TFIM V6.1, see "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 2572.

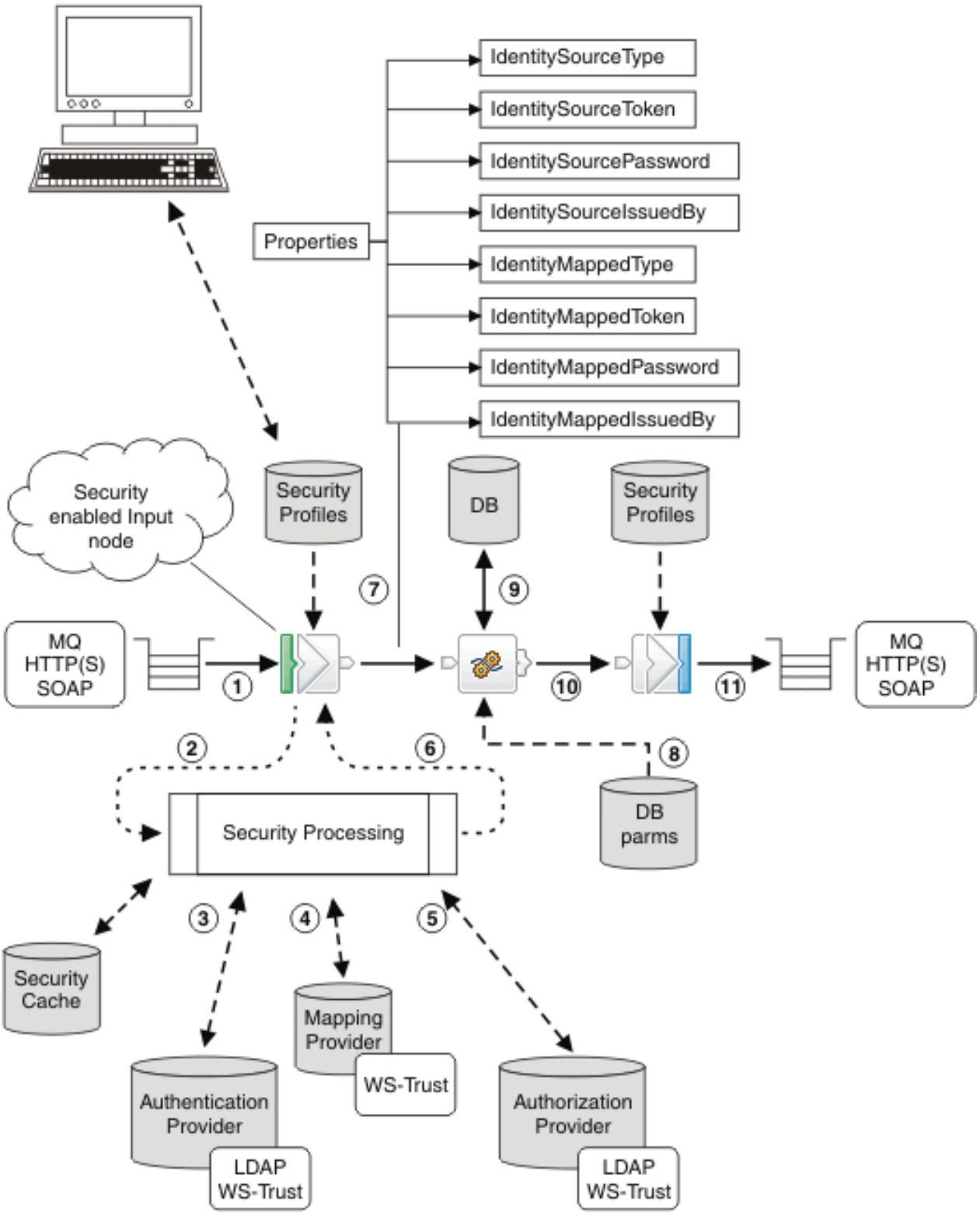
User-defined mapping

When you develop a message flow, you can implement a custom token mapping to be used for identity propagation. For example, you can implement a custom token mapping using a compute node (which can be a Compute or JavaCompute node) following the input node. In the compute node, you can read the source identity values from the Properties folder, process them, then write the new identity values to the mapped identity fields. If there is no identity provided in the message, you can still use a compute node to insert some identity credentials into the mapped identity fields. The mapped identity fields are then used in place of the source identity fields by subsequent nodes. Any security operations that are configured on the input node are performed using the source identity, before you can create a new identity in the mapped identity fields (by using the compute node). However, you can include a SecurityPEP node after your compute node has created a new mapped identity.

Invoking message flow security using a security enabled input node

You can invoke the message flow security manager by configuring a security enabled input node.

The following diagram shows an example message flow and gives an overview of the sequence of events that occur when an input message is received by a security enabled input node in the message flow:



For information about the security-enabled message flow nodes, see "Message flow security overview" on page 2550.

The following steps explain the sequence of events that occur when a message arrives at a security enabled input node in the message flow. The numbers correspond to those in the preceding diagram:

1. When a message arrives at a security enabled input node (MQ, HTTP, or SOAP), the presence of a security profile associated with the node indicates whether message flow security is configured.

SOAP nodes can implement some WS-Security capabilities without using the integration node's security manager; for more information, see “Implementing WS-Security” on page 2652. The integration server's security manager is called to read the profile, which specifies the combination of propagation, authentication, authorization, and mapping to be performed with the identity of the message. If an external security provider (Policy Decision Point or PDP) is to be used for one or more of these functions, the security profile specifies details of the provider to be used. If locally-stored credentials are to be used for authentication, the profile sets `Local` authentication and specifies the local credentials alias for the credentials in the independent integration server's vault. For more information, see “Authenticating incoming requests by using credentials stored in the vault” on page 2597.

You can create security profiles by using the Policy editor, as described in “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327. You then deploy the policy to the integration server where your associated message flows are deployed.

2. If a security profile is associated with the node or message flow, the security enabled input node extracts the identity information from the input message (based on the configuration on the node's **Security** properties page) and sets the Source Identity elements in the Properties folder. If the security tokens cannot be successfully extracted, a security exception is raised.

   If you require a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity), you must also define and specify an appropriate policy set and bindings for the relevant token profile. For more information, see “Policy sets” on page 2662.

   For HTTP and MQ input nodes, the **Security** properties page is used to configure the extraction of the identity. This defaults to *Transport Default*. For example, an HTTPInput node extracts a username and password from the HTTP BasicAuth header. The **Security** properties page allows the Identity token type and its location to be explicitly configured to control the extraction. This source identity information can be in a message header, the message body, or both.

   For information about the tokens that are supported by each node, see “Identity” on page 2554.

3. If authentication is specified in the security profile, the security manager either calls the configured security provider to authenticate the identity, or, if `Local` authentication has been configured, it authenticates the identity against the credentials that are held in the integration server's vault. A failure results in a security exception being returned to the node. The security providers that are supported by IBM Integration for authentication are LDAP, a WS-Trust v1.3 compliant Security Token Service (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the authentication result, which enables subsequent messages (with the same credentials) arriving at the message flow to be completed with the cached result, provided that it has not expired.

   For information about configuring an independent integration server to enforce authentication on all HTTP and SOAP input nodes, see “Configuring authentication for all HTTP and SOAP input nodes in an independent integration server” on page 2598.

4. If identity mapping is specified in the security profile, the security manager calls the configured security provider to map the identity to an alternative identity. A failure results in a security exception being returned to the node. Otherwise, the mapped identity information is set in the Mapped Identity elements in the Properties folder.

   The security providers that are supported by IBM Integration for identity mapping are a WS-Trust V1.3 compliant Security Token Service (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the result of the identity mapping.

   Alternatively, you can use a SecurityPEP node at any point in the message flow to map the identity that was authenticated at the security enabled input node. For more information, see “Invoking message flow security using a SecurityPEP node” on page 2569.

5. If authorization is specified in the security profile, the security manager calls the configured security provider to authorize that the identity (either mapped or source) has access to this message flow. A failure results in a security exception being returned to the node.

   The security providers that are supported by IBM Integration for authorization are LDAP, a WS-Trust V1.3 compliant Security Token Service (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the authorization result.

   Alternatively, you can use a SecurityPEP node at any point in the message flow to authorize the identity that was authenticated at the security enabled input node. For more information, see "Invoking message flow security using a SecurityPEP node" on page 2569.
6. When all security processing is complete, or when a security exception is raised by the message flow security manager, control returns to the input node.

   When a security exception is returned to the security enabled input node, it performs the appropriate transport handling and ends the message flow transaction. For example, an HTTPInput node returns an HTTP header with a 401 HTTP response code, without propagating to an output terminal. A SOAPInput node returns a SOAP Fault, reporting the security exception. Alternatively, if the `Treat security exceptions as normal` property is set on the security enabled input node, a security exception is propagated to the node's Failure terminal. The security enabled input node propagates to its Out terminal only if all the configured operations in the associated security profile complete successfully.
7. The message, including the Properties folder and its source and mapped identity information, is propagated down the message flow.
8. At subsequent nodes in the message flow an identity might be required to access a resource such as a database. The identity used to access such a resource is a proxy identity, an identity configured for the specific resource by using the **mqsisetdbparms** command.
9. When you are developing a message flow you can use the identity fields in the Properties folder for application processing (for example, identity-based routing or content building based on identity). Also, as an alternative to invoking mapping through a WS-Trust V1.3 enabled STS (such as TFIM V6.2) or TFIM V6.1, you can set the mapped identity fields in a compute node, such as a Compute, JavaCompute, or Mapping node.
10. When the message reaches a security enabled output or request node (MQOutput, HTTPRequest, SOAPRequest, or SOAPAsyncRequest), a security profile (with propagation enabled) associated with the node indicates that the current identity token is to be propagated when the message is sent.

    If the security profile indicates that propagation is required, the mapped identity is used. If the mapped identity is not set, or if it has a token type that is not supported by the node, the source identity is used. If no identity is set, or if neither the mapped nor source identity has a token type that is supported by the node, a security exception is returned to the node.

    SOAP nodes also require the appropriate policy set and bindings for the token profile to be associated with the node.

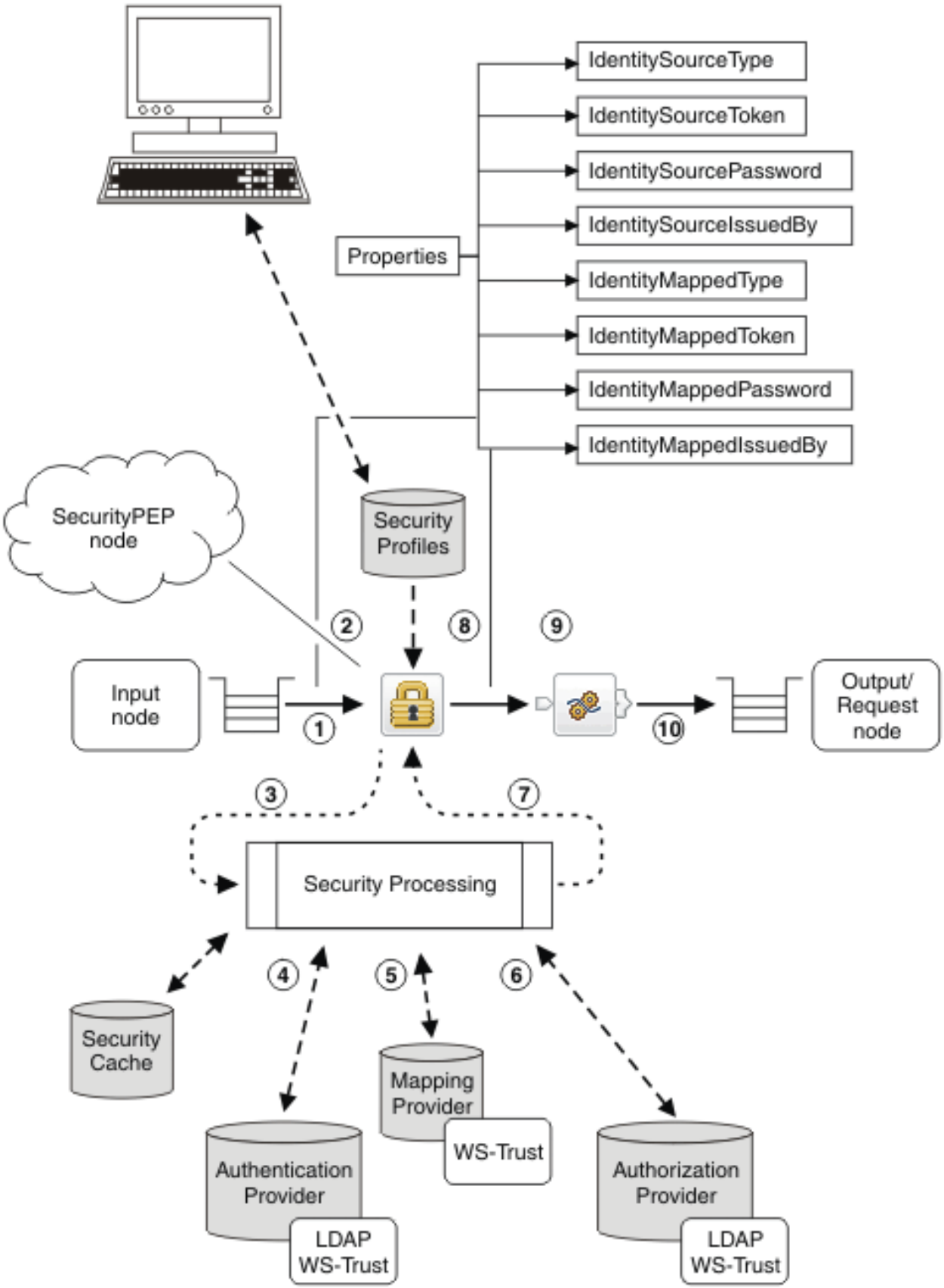
    If you want to include a security token in the message that is issued at an output node, and if the output node is not able to propagate that type of token, you can use a compute node (before the output node) to put the token from the properties tree into the relevant message location.

    For information about the tokens that are supported by each node, see "Identity and security token propagation" on page 2580.
11. The propagated identity is included in the appropriate message header when it is sent.

Invoking message flow security using a SecurityPEP node

You can invoke the message flow security manager at any point in a message flow, between an input node and an output or request node, by using a SecurityPEP node.

The following diagram shows an example message flow and gives an overview of the sequence of events that occur when an input message is received by an input node that is not security enabled (or that has no associated security profile) and is later processed by a SecurityPEP node in the message flow:


IdentitySourceType
IdentitySourceToken
IdentitySourcePassword
IdentitySourceIssuedBy
Properties
IdentityMappedType
IdentityMappedToken
IdentityMappedPassword
IdentityMappedIssuedBy
SecurityPEP node
Security Profiles
2
8
9
Input node
1
10
Output/ Request node
3
7
Security Processing
4
5
6
Security Cache
Mapping Provider
WS-Trust
Authentication Provider
LDAP WS-Trust
Authorization Provider
LDAP WS-Trust

The following steps explain the sequence of events that occur when a message arrives at an input node that is not security enabled (or that has no associated security profile). The numbers correspond to those in the preceding diagram:

1. You can use a SecurityPEP node at any point in a message flow between an input and an output or request node. The SecurityPEP node enables security to be applied in a message flow in the following situations:

   - When the message flow input node is not security enabled (for example, FileInput, TCPIPClientInput, SAPInput, and JMSInput nodes).
   - When the message flow input node is security enabled and might be configured to perform authentication operations, but the message flow is required to perform some routing or filtering before the business function being invoked is known; as a result, authorization needs to be performed later in the message flow logic.
   - When the message flow includes multiple output or request nodes, which require a specific identity mapping to be performed before each node, to obtain the appropriate security tokens for propagation.

   The message tree that is propagated into the SecurityPEP node includes the properties tree identity fields. These fields are empty, unless a security enabled input node (or a prior SecurityPEP node) has already extracted identity tokens, and possibly performed some security operations.

2. When a message arrives at a SecurityPEP node, the presence of a security profile associated with the node indicates whether message flow security is configured. The integration server's security manager is called to read the profile, which specifies the combination of propagation, authentication, authorization, and mapping to be performed with the identity of the message. If an external security provider (Policy Decision Point or PDP) is to be used for one or more of these functions, the security profile specifies details of the provider to be used. If locally-stored credentials are to be used for authentication, the profile sets `Local` authentication and specifies the local credentials alias for the credentials in the independent integration server's vault. For more information, see “Authenticating incoming requests by using credentials stored in the vault” on page 2597.

   You can create security profiles by using the Policy editor, as described in “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327. You then deploy the policy to the integration server where your associated message flows are running.

3. If a security profile is associated with the SecurityPEP node or message flow, the node extracts the identity information from the message tree based on the node configuration and sets the Source Identity elements in the Properties folder. If the node sets a token type of *Current token*, the existing identity tokens in the Mapped Identity properties fields are used (if they exist); if there are no identity tokens in the Mapped Identity properties fields, the tokens in the Source Identity properties fields are used. If the security tokens cannot be successfully extracted, a security exception is raised and propagated to the failure terminal (if wired).

4. If authentication is specified in the security profile, the security manager either calls the configured security provider to authenticate the identity, or, if `Local` authentication has been configured, it authenticates the identity against the credentials that are held in the integration server's vault. A failure results in a security exception being returned to the node. The external security providers that are supported by IBM App Connect Enterprise for authentication are a LDAP, WS-Trust v1.3 compliant Security Token Service (such as TFIM V6.2), and TFIM V6.1.

   A security cache is provided for the authentication result, which enables subsequent messages (with the same credentials) arriving at the message flow to be completed with the cached result, provided that it has not expired.

5. If identity mapping is specified in the security profile, the security manager calls the configured security provider to map the identity to an alternative identity. A failure results in a security exception

being returned to the node. Otherwise, the mapped identity information is set in the Mapped Identity elements in the Properties folder.

The security providers that are supported by IBM App Connect Enterprise for identity mapping are a WS-Trust V1.3 compliant Security Token Service (such as TFIM V6.2) and TFIM V6.1.

A security cache is provided for the result of the identity mapping.

6. If authorization is specified in the security profile, the security manager calls the configured security provider to authorize that the identity (either mapped or source) has access to this message flow. A failure results in a security exception being returned to the node.

   The security providers that are supported by IBM App Connect Enterprise for authorization are LDAP, a WS-Trust V1.3 compliant Security Token Service (such as TFIM V6.2) and TFIM V6.1.

   A security cache is provided for the authorization result.

7. When all security processing is complete, or when a security exception is raised by the message flow security manager, control returns to the SecurityPEP node.

   When a security exception is returned to the SecurityPEP node, the exception is either propagated to the failure terminal if it is connected, or returned to the preceding node as a recoverable exception. The SecurityPEP node propagates to its Out terminal only if all the configured operations in the associated security profile complete successfully.

8. The message, including the populated Properties folder and its source and mapped identity information, is propagated down the message flow.

9. When you are developing a message flow, you can use the identity fields in the Properties folder for application processing (for example, identity-based routing or content building based on identity). If the identity is to be propagated in an outbound message from an output or request node that does not support propagation of the token, you can use a compute node (including a Compute, JavaCompute, or Mapping node), to move the identity token into the required transport header or message body location.

10. When the message reaches an output node, a security profile associated with the node can indicate whether an identity is to be taken from the Properties folder and propagated when the message is sent. Only specific transport nodes can propagate tokens that are the default for the transport; any other token type must be handled by a compute node, as described above.

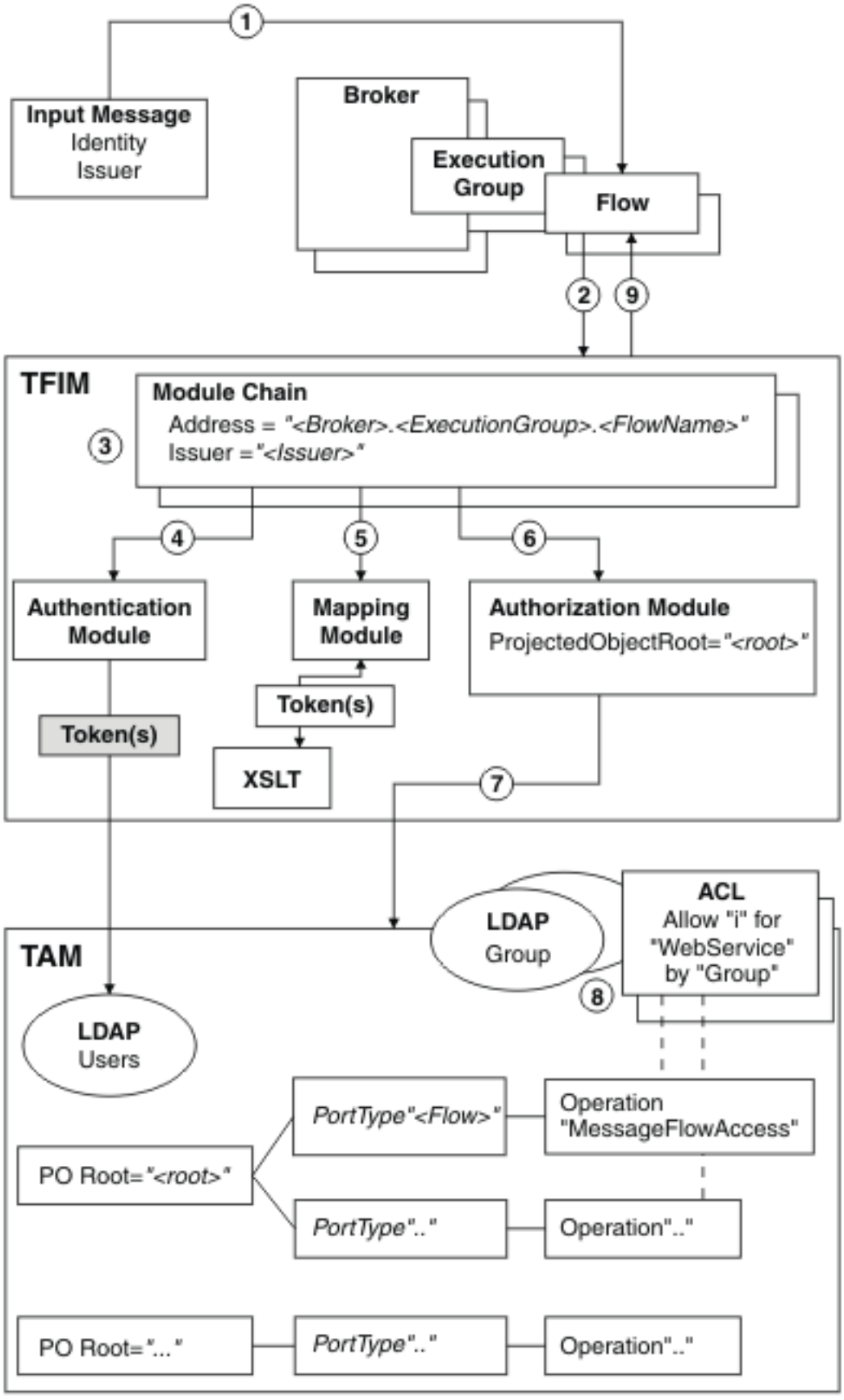
Authentication, mapping, and authorization with TFIM V6.1 and TAM

Use IBM App Connect Enterprise, Tivoli Federated Identity Manager (TFIM) V6.1, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

Note: Support for TFIM V6.1 is included for compatibility with previous versions of IBM App Connect Enterprise. If possible, upgrade to TFIM V6.2 and refer to the information in "Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574.

IBM App Connect Enterprise makes a single TFIM WS-Trust call for an input node that is configured with a TFIM security profile, which means that a single module chain must be configured to perform all the required authentication, mapping, and authorization operations.

The following diagram shows the configuration of IBM App Connect Enterprise, TFIM, and TAM to enable authentication, mapping, and authorization of an identity in a message flow:



The numbers in the preceding diagram correspond to the following sequence of events:

1. A message enters a message flow.
2. A WS-Trust request is issued by the integration node, with these properties:
   - RequestType = Validate
   - Identity = Token(s) from input message
   - Issuer = Issuer from input message
   - AppliesTo Address = *"Broker.IntegrationServer.FlowName"*
   - PortType = *"FlowName"*
   - Operation = "MessageFlowAccess"
3. TFIM selects a module chain to process the WS-Trust request, based on the AppliesTo Address and Issuer properties of the request.
4. A module chain can perform authentication if it includes a module (such as a UsernameTokenSTSModule or X509STSModule) in `validate` mode.
5. A module chain can perform mapping by using an XSLTransformationModule in `mapping` mode to manipulate the identity information.
6. A module chain can perform authorization by using an AuthorizationSTSModule in `other` mode. The module chain must be configured with a *Protected Object Root* value.
7. The AuthorizationSTSModule performs the authorization check by making a request to TAM with these properties:
   - Action = "i" (invoke)
   - Action Group = "WebService"
   - Protected Object = "*ProtectedObjectRoot.FlowName.MessageFlowAccess"*

     where *"i"* and *"WebService"* are default values used by an AuthorizationSTSModule; and *FlowName* and *MessageFlowAccess* are the WS-Trust request PortType and Operation values.
8. TAM processes the authorization request by:
   a. Finding the Access Control Lists (ACLs) associated with protected object *"<ProtectedObjectRoot>.<FlowName>.MessageFlowAccess"*.
   b. Checking whether or not the ACLs grant action "i" on action group "WebService" to the user (with the user either named directly, or by membership of a named group).
9. The WS-Trust reply is returned to the integration node. If this action is the result of a mapping request, the WS-Trust reply contains the mapped identity token.

Authentication, mapping, and authorization with TFIM V6.2 and TAM

You can use IBM App Connect Enterprise, Tivoli Federated Identity Manager (TFIM) V6.2, and Tivoli Access Manager (TAM) to control authentication, mapping, and authorization.

IBM App Connect Enterprise makes a single TFIM WS-Trust call for an input node or SecurityPEP node that is configured with a WS-Trust V1.3 STS security profile. As a result, a single module chain must be configured to perform all the required authentication, mapping, and authorization operations.

When you use a WS-Trust v1.3 STS for authentication, authorization, or mapping, a request is made to the trust service with the following parameters, which control the STS processing. If you are using TFIM V6.2, these parameters are used in the selection of the TFIM module chain:

Parameter	Value
RequestType	The type of request issued to the trust service. Valid values are: **`Issue`** This value can be specified when mapping is the only operation that is set to `WS-Trust V1.3 STS` in the security profile. It is not valid if `WS-Trust V1.3 STS` is specified for authentication or authorization. The namespace qualified value is `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue`, which shows in TFIM V6.2 as *Issue Oasis URI*. **`Validate`** This value must be set when the security profile also includes authentication or authorization (in addition to mapping) for the same `WS-Trust V1.3 STS` provider. The namespace qualified value is `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate`, which shows in TFIM V6.2 as *Validate Oasis URI*.
Issuer	This value is determined by the effective setting of the `IssuedBy` property on the **Basic** tab of the SecurityPEP node or the **Security** tab of the input node.

Parameter	Value

AppliesTo	This value is determined by the type of node:
	MQInput node with MQ binding:
	The IBM MQ IRI of the node's input queue; for example:
	`wmq://msg/queue/queue_name@queue_manager_name`
	HTTPInput, SOAPInput, or SOAPAsyncResponse node with HTTP binding:
	The endpoint URL; for example:
	`http://myflow/myInputNodePath`
	SecurityPEP node with a default (blank) WS-Trust AppliesTo address:
	The URN for the message flow that contains the node; for example:
	`urn:/integrationNodeName.integrationServerName.flowName`
	SecurityPEP node with WS-Trust AppliesTo address set on the Advanced tab of the node:
	The URI value configured in the property. This value is typically the URL of the target service that is used when you invoke a mapping operation to obtain the required token for the following request node; for example:
	`http://remotehost.ibm.com:9080/targetservice`
	You can also set the `AppliesTo service name` and `AppliesTo port type` properties on the **Advanced** tab of the node. The WS-Trust request includes these optional elements only when they are configured. These values are typically valid QNames; for example:
	`http://myservice.mycom.com:myservicename`
	When you set these properties in the SecurityPEP node, you must configure them in the TFIM module chain:
	• In the service name and port type TFIM properties, the information to the left of the colon must match the namespace URI of the WS-Addressing namespace that is used for the PortType and ServiceName elements in the WS-Trust request set by the integration node, which is:
	`http://www.w3.org/2005/08/addressing`
	• The information to the right of the colon in the service name and port type TFIM properties must match the value configured on the SecurityPEP node. You can also configure a regular expression in TFIM to specify a match.

This section describes an authorization configuration that you can use to perform the authorization operation with TFIM V6.2 and TAM.

In the security profile, set the TFIM V6.2 endpoint for the authorization operation. When you create a module chain to be used by a security enabled input node or SecurityPEP node, and resolved by *AppliesTo* information, you must include the TFIM TAMAuthorizationSTModule to invoke TAM authorization.

The TAMAuthorizationSTSModule requires the following TFIM STS universal user context attributes:

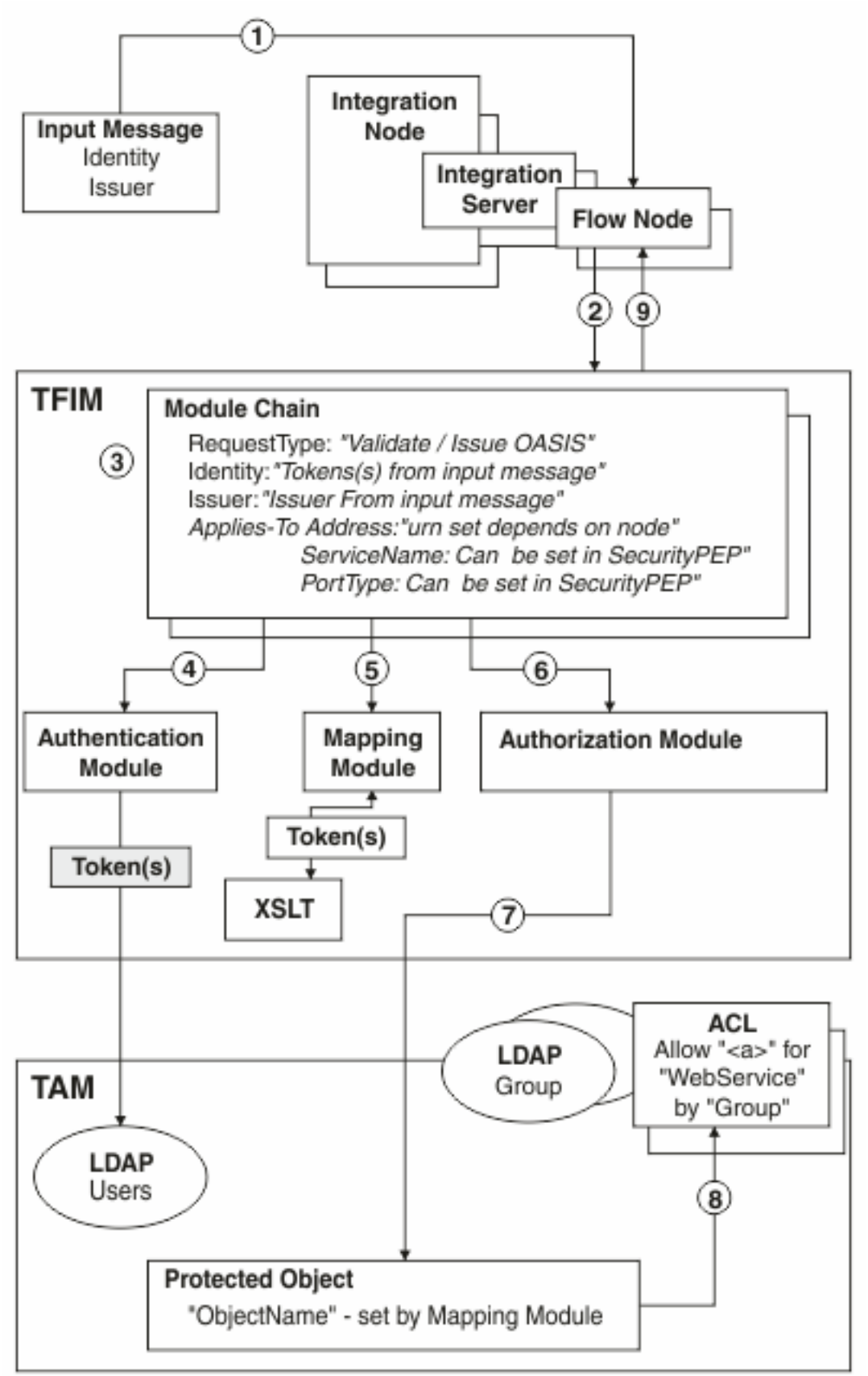
PrincipalName
The username to be authorized. This username must exist in your TAM user repository.

ObjectName
The TAM object name of the resource on which an authorization check is to be made. Typically this is derived from the *AppliesTo* information that is passed by the message flow security manager from the security enabled input node or SecurityPEP node.

Action
The TAM action to be authorized; for example, *x* (eXecute).

The TAM Access Control List (ACL), which determines the authorization decision, is located in the TAM protected object space using the path that is set on the *ObjectName* attribute of the TFIM STS universal user context input to the TAMAuthorizationSTSModule module.

The following diagram shows the configuration of IBM App Connect Enterprise, TFIM V6.2, and TAM to enable authentication, mapping, and authorization of an identity in a message flow:



The numbers in the preceding diagram correspond to the following sequence of events:

1. A message enters a message flow.

2. A WS-Trust request is issued by the integration node, with the *RequestType*, *Issuer*, and *AppliesTo* properties set.
3. TFIM selects a module chain to process the WS-Trust request, based on the *RequestType*, *Issuer*, and *AppliesTo* properties of the request.
4. A module chain can perform authentication if it includes a module in *Validate* mode that is appropriate to the token type that is being passed in the request from the message flow input message. For example, a Username and Password token can be authenticated using a UsernameTokenSTSModule .
5. The module chain must perform some mapping by using an XSLTransformationModule in mapping mode to manipulate the identity information and to provide the required context attributes in the TFIM stsuser object for use by subsequent modules.
6. A module chain can perform authorization in TAM by using the TAMAuthorizationSTSModule.
7. The TAMAuthorizationSTSModule performs the authorization check by making a request to TAM with these properties:
   - Action = *a* (where *a* is the stsuser context action attribute). For example, *x* for eXecute could be set using the following code:

     ```
     <stsuuser:ContextAttributes>
         <!-- Action -->
         <stsuuser:Attribute name="Action" type="urn:ibm:names:ITFIM:stsmodule:tamazn">
           <stsuuser:Value>x</stsuuser:Value>
         </stsuuser:Attribute>
       </stsuuser:ContextAttributes>
     ```

   - Action Group = *WebService*
   - Protected Object = *ProtectedObjectName* (where *ProtectedObjectName* is the stsuser context action attribute). For example, *x* for eXecute could be set using the following code:

     ```
     <stsuuser:ContextAttributes>
         <!-- ObjectName -->
         <stsuuser:Attribute name="ObjectName" type="urn:ibm:names:ITFIM:stsmodule:tamazn">
           <stsuuser:Value>ProtectedObjectName</stsuuser:Value>
         </stsuuser:Attribute>
       </stsuuser:ContextAttributes>
     ```

     Typically, *ProtectedObjectName* is set conditionally from the *AppliesTo* information in the request.
8. TAM processes the authorization request by:
   a. Finding the Access Control Lists (ACLs) associated with protected object *ProtectedObjectName*
   b. Checking whether the ACLs grant action *a* on action group *WebService* to the user (the user is named either directly or indirectly, through membership of a named group).
9. The WS-Trust reply is returned to the integration node. If this action is the result of a mapping request, the WS-Trust reply contains the mapped identity token.

Configuring the security cache

You can configure the security cache by specifying a value for the **cacheTimeout** property for an independent integration server by updating the `server.conf.yaml` configuration file or, for on an integration node, by using the **mqsichangeproperties** command.

About this task

To improve performance, the security manager uses a security cache. Entries are created in the security cache when a message flow with a security profile performs authentication, mapping, or authorization. The entries are valid for the length of time that is specified by the `cacheTimeout` property of the `SecurityCache` component after which the entries are marked as expired.

Procedure

- For an independent integration server:
  a) Open the configuration file for your integration server (`server.conf.yaml`) by using a YAML editor.

     If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid characters in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file. For more information about configuring an integration server, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
  b) Set the security cache timeout property, **cacheTimeout**, in the **SecurityCache** section of the `server.conf.yaml` configuration file.

     The default value is 60 seconds.
  c) When you have modified and saved the `server.conf.yaml` file, restart the integration server for the changes to take effect.

     For information about how to start an integration server, see "Starting an integration server" on page 250.
- For an integration node:
  a) Use the **mqsichangeproperties** command.

     For example, to change the value of the security cache timeout parameter, **cacheTimeout**, to 200 seconds, use the following command:

     ```
     mqsichangeproperties INODE -b SecurityCache -n cacheTimeout -v 200
     ```
  b) Restart the integration node for the change to take effect.

Identity and security token propagation

Identity and security token propagation enables the identity and security tokens (associated with each message) to be propagated throughout a message flow, and on to target applications through output or request nodes.

In an enterprise system, you can use different physical identities or security tokens (such as user names, certificates, and SAML assertions) to represent a single logical identity through different parts of the enterprise. The propagation of an identity or security token ensures that the logical identity is kept throughout the system by mapping between the various physical forms as necessary. For example, a message might enter the system using a certificate, but a user name token might be required for server processing of the message. Identity mapping is used to convert from the certificate to the `Username` token, and identity propagation ensures that the mapped identity is placed in the correct place for the outbound transport.

When an output or request node propagates an identity, the mapped identity is used. If the mapped identity is not set, or if it has a token type that is not supported by the node, the source identity is used. You can also configure a fixed identity by using the **mqsisetdbparms** command; see "Configuring a message flow for identity propagation" on page 2613. If no identity is set, or if neither the mapped nor source identity has a token type that is supported by the node, a security exception is thrown by the node.

The following output nodes support identity propagation:

- CICSRequest
- HTTPRequest
- IMSRequest
- MQOutput
- SAPRequest
- SOAPAsyncRequest

- SOAPRequest
- SiebelRequest
- RESTRequest
- RESTAsyncRequest

The following table shows the support that is provided by the message flow security manager for the propagation of the different types of security token. For more information about these security tokens, see “Identity” on page 2554.

Table 91. Support for security token types - token propagation

Token type (format)	**Integration node security manager support**	**Token propagated in**
Username	Username tokens are supported for propagation by the following nodes: • CICSRequest • HTTPRequest • IMSRequest • MQOutput • RESTAsyncRequest • RESTRequest • SOAPRequest	**CICS** The request security credentials **HTTP** BasicAuth header **IMS** The request security credentials **IWA** The request security credentials **MQ** MQMD.UserIdentifier transport header
Username and password	Username and password tokens are supported for propagation by the following nodes: • CICSRequest • HTTPRequest • IMSRequest • MQOutput • RESTAsyncRequest • RESTRequest • SAPRequest • SOAPAsyncRequest • SOAPRequest • SiebelRequest	**CICS** The request security credentials **HTTP** BasicAuth header **IMS** The request security credentials **IWA** The request security credentials **MQ** MQMD.UserIdentifier transport header **SAP** The request security credentials **SOAP** • BasicAuth header, if there is no policy set and binding • SOAP header, if a policy set and binding sets the Username token profile • Kerberos client credentials, if a policy set and binding sets the Kerberos token profile **SIEBEL** The request security credentials

Table 91. Support for security token types - token propagation (continued)		
Token type (format)	**Integration node security manager support**	**Token propagated in**
SAML assertion	SAML tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the SAML token profile
X.509 certificate	X.509 tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the X.509 binary token profile
LTPA v2 token	LTPA v2 tokens are supported for propagation by the following nodes: • SOAPRequest • SOAPAsyncRequest	SOAP header when a policy set and binding sets the LTPA token profile
Universal WSSE token	Universal WSSE tokens are not supported for propagation by any node.	

For information about how to configure a message flow to propagate a message identity, see “Configuring a message flow for identity propagation” on page 2613. For more information about how one physical identity is converted to another, see “Identity mapping” on page 2564.

Security identities for the integration server connecting to external systems

You can access external systems from the message flows that you deploy to your integration server, and you must therefore consider the steps that you might want to take to secure that access.

You can set a user ID and password that you want the integration server to use for access to an external system or database by using the **mqsisetdbparms** command on a policy .

After you have defined user IDs, you must authorize those IDs so that the integration server can access your databases from deployed message flows. See the documentation for your database provider to authorize your integration node user ID.

To see how to set security authorization for a message flow node, refer to the table of properties for that specific node.

IBM App Connect Enterprise does not provide special commands to administer databases. Discuss your database security requirements with the database administrator for the database manager that you are using, or refer to the documentation provided by your database supplier.

Security exception processing

A security exception is raised when a message flow security failure occurs during security processing in an input node or SecurityPEP node.

Security exceptions are processed in a different way from other errors on the input node. An error is typically caught on the input node and routed down the Failure terminal for error processing, but security exceptions are not processed in the same way. By default, the integration node does not allow security exceptions to be caught within the message flow, but backs the message out or returns an error (as in the case of HTTP). Security exceptions in input nodes are managed in this way to prevent a security denial of service attack filling the logs and destabilizing the system.

However, security exceptions in SecurityPEP nodes are managed in a different way. If a security operation fails in a SecurityPEP node, a security exception is raised, wrapped in a normal recoverable exception, which invokes the error handling that is provided by the message flow.

If you have designed the message flow to be in a secure area and you want to explicitly perform processing of security exceptions, you can select the **Treat Security Exceptions as normal exceptions** property on the MQInput or HTTPInput nodes. This property causes security exceptions to be processed in the same way as other exceptions in the message flow.

If you associate the Default Propagation security profile with an output or request node, the token type of the mapped or source security token must be the same as the transport default for that node; otherwise, a security exception occurs. For example, for an MQOutput node, the token type must be *Username*, for an HTTPRequest node, the token type must be *Username + Password*, and for a SOAPRequest node, the token type must be the type that is defined in either the policy set and binding or the transport binding.

For information on how to diagnose the causes of security exceptions, see "Diagnosing security problems" on page 2630.

Setting up message flow security

Set up security on a message flow to control access based on the identity of a message passing through the message flow.

About this task

You can configure the integration server to perform end-to-end processing of an identity carried in a message through a message flow. Administrators can configure security at message flow level, controlling access based on the identity flowed in a message. This security mechanism is independent of both the transport and the message format.

To set up security for a message flow, perform the tasks described in the following topics:

Procedure

1. "Creating a security profile" on page 2584
2. "Configuring the extraction of an identity or security token" on page 2590
3. "Configuring identity authentication and security token validation" on page 2593
4. "Configuring identity mapping" on page 2603
5. "Configuring authorization" on page 2605
6. "Configuring a message flow for identity propagation" on page 2613
7. Securing access to external systems
8. "Diagnosing security problems" on page 2630

What to do next

If the message flow is a web service implemented by using the "SOAP nodes" on page 811, and the identity is to be taken from the "WS-Security" on page 2650 header tokens, you must also create appropriate "Policy sets" on page 2662 and bindings, then configure them on the relevant SOAP nodes (in addition to the security profile). See "Associating policy sets and bindings with message flows and nodes" on page 2672.

To work with an identity, you must configure the policy sets and bindings for the relevant capabilities:

- To work with a Username and Password identity, configure the policy sets and bindings for "Username token capabilities" on page 2675.

- To work with an X.509 Certificate identity, configure the policy sets and bindings for "X.509 certificate token capabilities" on page 2677.

  In the policy set binding, the `Certificates` mode of the X.509 certificate authentication token must be set as `Trust Any` (rather than `Trust Store`), so that the certificate is passed to the security provider defined by the security profile. Setting *Trust Store* causes the certificate to be validated in the local trust store.
- To work with a SAML assertion, configure the policy sets and bindings for "SAML token capabilities" on page 2680.
- To work with an LTPA token, configure the policy sets and bindings for "LTPA token capabilities" on page 2684.
- To work with a Kerberos ticket, configure the policy sets and bindings for "Kerberos token capabilities" on page 2682.

For more information, see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel.

Creating a security profile

You can create a security profile for use with an external security provider such as Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2. Support is also provided for TFIM V6.1, for compatibility with previous versions of IBM App Connect Enterprise. You can also create a security profile for authenticating against credentials that are held locally in the integration server's vault.

About this task

Before you can enable security on a node or a message flow, you need to create a security profile that defines the security operations that you want to perform.

You can create a security profile for use with external security providers to provide the required security enforcement and mapping. You can configure the security profile to use different security providers for different security functions; for example, you might use LDAP for authentication and WS-Trust V1.3 STS for mapping and authorization. You can also create a security profile for authenticating against credentials that are held locally in the integration server's vault, as described in "Authenticating incoming requests by using credentials stored in the vault" on page 2597.

You can create the security profile by using the Policy editor (see Security Profiles policy (SecurityProfiles)).

If you want to extract and propagate an identity without security enforcement or mapping, you can use the supplied security profile called Default Propagation. The Default Propagation profile is a predefined profile that requests only identity propagation.

To create a security profile, see:

- "Creating a security profile for LDAP" on page 2584
- "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588
- "Creating a security profile for TFIM V6.1" on page 2589
- "Creating a security profile for using locally-stored credentials" on page 2590

Creating a security profile for LDAP

Create a security profile for use with Lightweight Directory Access Protocol (LDAP) or Secure LDAP (LDAPS), by using the Policy editor.

Before you begin

Ensure that you have an LDAP server that is LDAP Version 3 compliant, for example:

- IBM Tivoli Directory Server

- Microsoft Active Directory
- OpenLDAP.

About this task

If your LDAP directory does not permit login by unrecognized user IDs, and does not grant search access rights on the subtree, you must also set up a separate authorized login ID that the integration node can use for the search. For information on how to do this, see “Configuring authorization with LDAP” on page 2606 or “Authenticating incoming requests with LDAP” on page 2593.

Creating a security profile

About this task

You can create a Security Profiles policy that uses LDAP for authentication, authorization, or both. The security profile ensures that each message has an authenticated ID and is authorized for the message flow.

Procedure

1. Create a policy and choose the Security Profiles type (see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327).
2. Set appropriate values for the properties (see Security Profiles policy (SecurityProfiles)).

   You must enclose the LDAP URL (which contains commas) with escaped double quotation marks (\" and \") so that the URL commas are not confused with the comma separator of the value parameter.

   If the LDAP URL includes an element name with a space, in this case `cn=All Sales`, the set of values after the -v flag must be enclosed by double quotation marks, (")

   You can define the security-specific parts of the command in the following way:

   a) Set the **`authentication`** to *LDAP*.
   This ensures that the incoming identity is validated.

   b) Set the **`authenticationConfig`** using the following syntax:

   ```
   ldap[s]://server[:port]/baseDN[?[uid_attr][?[base|sub]]]
   ```

   For example:

   ```
   ldap://ldap.acme.com:389/ou=sales,o=acme.com
   ```

```
ldaps://localhost:636/ou=sales,o=acme?cn?base
```

ldap:
(Required) Fixed protocol string.

s:
(Optional) Specifies whether SSL should be used. Default is not to use SSL.

server:
(Required) The name or IP address of the LDAP server to contact.

port:
(Optional) The port to connect to. Default is 389 (non-SSL). For LDAP servers with SSL enabled, the port is typically 636.

baseDN
(Required) String defining the base distinguished name (DN) of all users in the directory. If users exist in different subtrees, specify a common subtree under which a search on the username uniquely resolves to the required user entry, and set the *sub* attribute.

uid_attr:
(Optional) String defining the attribute to which the incoming username maps, typically uid, CN, or email address. Default is *uid*.

base|sub:
(Optional) Defines whether to perform a base or subtree search. If `base` is defined, the authentication is faster because the DN of the user can be constructed from the *uid_att*, *username*, and *baseDN* values. If *sub* is selected, a search must be performed before the DN can be resolved. Default is *sub*.

c) (Optional) To specify that you want the security manager to reject a user name during authentication if the user name has an empty password token, set **rejectBlankpassword** to TRUE.

The default is FALSE, which means that a user name is authenticated against the LDAP server even if it has an empty password token.

d) (Optional) To specify the way that the password is displayed in the properties folder, set **passwordValue** to one of the following values:

PLAIN
The password is displayed in the Properties folder as plain text.

OBFUSCATE
The password is displayed in the Properties folder as base64 encoding.

MASK
The password is displayed in the Properties folder as four asterisks (****).

e) Set **authorization** to *LDAP*.

This ensures that the incoming identity is checked for group membership in LDAP.

f) Set **authorizationConfig** using the following syntax:

```
ldap[s]://server[:port]/groupDN[?[member_attr]
[?[base|sub][?[x-userBaseDN=baseDN,
x-uid_attr=uid_attr]]]]
```

For example:

```
ldap://ldap.acme.com:389/cn=All Sales,ou=acmegroups,
o=acme.com?uniquemember?sub?x-userBaseDN=ou=sales%2co=ibm.com,
```

```
x-uid_attr=emailaddress
```

ldap:
(Required) Fixed protocol string

s:
(Optional) Specifies whether SSL is used. Default is not to use SSL.

server:
(Required) The name or IP address of the LDAP server to contact.

port:
(Optional) The port to connect to. Default is 389 (non-SSL). For LDAP servers with SSL enabled, the port is typically 636.

groupDN
(Required) Fully defined distinguished name (DN) of the group in which users must be members to be granted access.

You can specify multiple group DNs by using the following formats:

- ```
  |(groupDN1)(groupDN2)(groupDN3)...(groupDNn)
  ```

  Authorization succeeds if the user is a member of any of the groups that are specified. For example:

  ```
  ldap://ldap.acme.com:389/|(cn=UK Sales,ou=acmegroups,o=acme.com)(cn=US
   Sales,ou=acmegroups,o=acme.com)?uniquemember?sub?x-userBaseDN=ou=sales
  %2co=ibm.com,x-uid_attr=emailaddress
  ```

- ```
  &(groupDN1)(groupDN2)(groupDN3)...(groupDNn)
  ```

  Authorization succeeds if the user is a member of all the groups that are specified. For example:

  ```
  ldap://ldap.acme.com:389/&(cn=UK Sales,ou=acmegroups,o=acme.com)(cn=US
   Sales,ou=acmegroups,o=acme.com)?uniquemember?sub?x-userBaseDN=ou=sales
  %2co=ibm.com,x-uid_attr=emailaddress
  ```

- UNIX The UNIX command line interprets the | and & characters as special characters. You must enclose multiple group DNs in double quotation marks, as shown in the following example:

  ```
  ldap://ldap.acme.com:389/"|(cn=UK Sales,ou=acmegroups,o=acme.com)(cn=US
   Sales,ou=acmegroups,o=acme.com)"?uniquemember?sub?x-userBaseDN=ou=sales
  %2co=ibm.com,x-uid_attr=emailaddress
  ```

member_attr:
(Optional) The attribute of the group used to filter the search. Default is to look for both **`member`** and **`uniquemember`** attributes.

The following options are required only if authentication has not preceded the authorization, and if the authentication configuration string has not been specified. If the authentication configuration

string has been specified, the following parameters are ignored and those provided by the **baseDN**, **uid_attr**, and **[base|sub]** for authentication are used instead:

base|sub:
Defines whether to perform a base or subtree search. If `base` is defined, the authentication is faster because the DN of the user can be constructed from uid_att + username + baseDN. If `sub` is selected, a search must be performed before the DN can be resolved. Default is `sub`.

baseDN
String defining the base distinguished name of all users in the directory. Must be preceded by the string `x-userBaseDN`. Any commas in the BaseDN must be rendered as `%2c`.

x-uid_attr:
String defining the attribute to which the incoming username should map, typically uid, CN, or email address. Default is uid. Must be preceded by the string `x-uid_attr`.

When you submit the command from a batch (.bat) file or command (.cmd) file, if the LDAP URL includes an extension with LDAP URL "percent hex hex" escaped characters (for example, a comma replaced by `%2c`, or a space replaced by `%20`), the percent signs must be escaped from the batch interpreter with an extra percent sign (`%%`).

The selected group must be defined on the LDAP server, and all of the required users must be members of the group.

Creating a security profile for WS-Trust V1.3 (TFIM V6.2)

You can create a security profile for a WS-Trust V1.3 compliant Security Token Service (STS), for example, Tivoli Federated Identity Manager (TFIM) V6.2, for any combination of the following security operations: authentication, authorization, and mapping.

Creating a profile

About this task

To create a security profile that uses a WS-Trust V1.3 compliant STS, set the configuration parameter of a Security Profiles policy to the full URL of the STS. The URL must consist of the transport scheme, host name, port, and path. For TFIM V6.2 WS-Trust V1.3 endpoint, the path is `/TrustServerWST13/services/RequestSecurityToken`. For example:

```
http://stsserver.mycompany.com:9080/TrustServerWST13/services/RequestSecurityToken
```

Procedure

1. Create a policy and choose the Security Profiles policy type (see “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327).
2. Set the `Mapping` property to `WS-Trust v1.3 STS`.
3. Set the Mapping configuration property to the full URL of the STS.

   To specify that you want the security manager to reject a user name during authentication if the user name has an empty password token, set **`rejectBlankpassword`** to TRUE. The default is FALSE,

which means that a user name is authenticated against the WS-Trust server even if it has an empty password token.

To specify the way that the password is displayed in the properties folder, set **passwordValue** to one of the following values:

PLAIN
The password is displayed in the Properties folder as plain text.

OBFUSCATE
The password is displayed in the Properties folder as base64 encoding.

MASK
The password is displayed in the Properties folder as four asterisks (****).

If the URL specifies an address beginning with `https://`, an SSL secured connection is used for requests to the WS-Trust v1.3 server.

In addition to specifying the security profile URL as an address beginning with `https://`, you can configure the following advanced parameters, by setting integration node environment variables:

MQSI_STS_SSL_PROTOCOL
The version of the SSL protocol to be used. Valid values are:

- SSL
- SSLv3

  Note: SSLv3 is disabled by default in IBM App Connect Enterprise 12.0, because SSLv3 is no longer considered secure.
- TLS

The initial value is TLS.

MQSI_STS_SSL_ALLOWED_CIPHERS
A space-separated list of the encryption ciphers that can be used. For a list of all the cipher suites that are supported by IBM App Connect Enterprise, see the Java product information for your operating system. For operating systems that use IBM Java, see Appendix A of the IBM JSSE2 Guide: http://www.ibm.com/developerworks/java/jdk/security/60/secguides/jsse2Docs/JSSE2RefGuide.html

MQSI_STS_REQUEST_TIMEOUT
The STS request timeout, specified in seconds. The initial value is `100`. For information about providing environment variables to the integration node, see "Setting up a command environment" on page 64.

MQSI_STS_REQUEST_ONBEHALFOF
If you set this environment variable to `enabled`, the WS-trust requests include the optional field `wst:OnBehalfOf`. This field is required when the STS server is secured and needs credentials/authentication tokens before it process the request.

If WS-Trust v1.3 STS is selected for more than one operation (for example, for authentication and mapping), the WS-Trust v1.3 server URL must be identical for all the operations, and is therefore specified only once.

Creating a security profile for TFIM V6.1

You can create a security profile for Tivoli Federated Identity Manager (TFIM) V6.1 for any combination of the following functions: authentication, authorization, and mapping.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of IBM App Connect Enterprise. If possible, upgrade to TFIM V6.2 and follow the instructions in "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

About this task

Create a Security Profiles policy that uses TFIM V6.1, and set the configuration parameter to the URL of the TFIM server. For example: `http://tfimserver.mycompany.com:9080`

Procedure

1. Create a policy and choose the Security Profiles type (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Set the `Mapping` property to `TFIM`.
3. Set the `Mapping configuration` property to the URL of the provider, for example `http://tfimserver.mycompany.com:9080`.

   If the URL specifies an address beginning with `https://`, an SSL secured connection is used for requests to the TFIM server.

   If TFIM is selected for more than one operation (for example, for authentication and mapping), the TFIM server URL must be identical for all the operations, and is therefore specified only once.

Creating a security profile for using locally-stored credentials

You can create a security profile for authentication against credentials stored locally in the integration server's vault.

About this task

Create a Security Profiles policy that uses locally-stored credentials for authentication, by completing the following steps:

Procedure

1. Create a Security Profiles policy in a policy project (for example, `SecPolicy/BasicAuthLocal.policyxml`), as described in "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.
2. Set the **`Authentication`** property to `Local`.
3. Set the **`AuthenticationConfiguration`** property to the local credential alias of the credential that is stored in the integration server's vault. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<policies>
  <policy policyType="SecurityProfiles" policyName="LocalAuth"
 policyTemplate="SecurityProfiles">
    <authentication>Local</authentication>
    <authenticationConfig>LocalCredentialsAlias</authenticationConfig>
  </policy>
</policies>
```

Configuring the extraction of an identity or security token

You can configure the SecurityPEP node or security enabled input nodes to extract the identity or security token from a message and store it in the properties tree identity fields, enabling it to be processed throughout the message flow and propagated at output or request nodes.

Before you begin

Check that an appropriate security profile exists or create a new security profile. See "Creating a security profile" on page 2584.

About this task

If an input node or SecurityPEP node is associated with a profile that specifies a security operation (authentication, mapping, or authorization), or specifies propagation as enabled, the node can retrieve an identity or security token from the message bit stream.

- An MQInput node, with the `Identity token type` security property set to `Transport default`, retrieves the UserIdentifier element from the message descriptor (MQMD) and puts it into the Identity Source Token element of the Properties folder. At the same time, it sets the Identity Source Type element to `username` and the Identity Source Issued By element to `MQMD.PutApplName` (the put application name).
- An HTTPInput node, with the `Identity token type` security property set to `Transport default`, retrieves the BasicAuth header from the HTTP request, decodes it, and puts it into the Identity Source Token and Password elements in the Properties folder. At the same time, it sets the Identity Source Type element to `username + Password` and the Identity Source Issued By element to the HTTP header `UserAgent` property.
- A SOAPInput node retrieves the appropriate tokens as defined by the configured WS-Security policy sets and bindings, or (if they are not set), the transport binding determines the token type; for example, HTTP transport is BasicAuth. The SOAPInput node then populates the identity source fields in the Properties folder with the retrieved tokens. With a Kerberos policy set and bindings, the token type is a Username containing the Service Principal Name (SPN) from the Kerberos ticket.
- A SecurityPEP node, with the `Identity token type` property set to `Current token`, can use the token that has been extracted by an upstream input or SecurityPEP node and stored in the Properties folder.

In some cases, the information extracted from the transport headers is not set or is insufficient to perform authentication or authorization. For example, for authentication to occur, a Username + Password type token is required; however, with WebSphere MQ, only a username is available, which means that the incoming identity has to be trusted. However, you can increase security by applying transport-level security using WebSphere MQ Extended Security Edition.

If the transport header cannot provide the required identity credentials, the information must be provided as part of the body of the incoming message. To enable the identity information to be taken from the body of the message, you must specify the location of the information by using either the **Security** tab on the HTTP, MQ, and SCA input nodes or the **Basic** tab on the SecurityPEP node, or by configuring the required policy set and bindings WS-Security profile on the SOAP node. A SOAP node with a Kerberos policy set and bindings extracts a Username token containing the Service Principal Name (SPN) of the Kerberos ticket.

Procedure

1. In **Identity Token Type**, specify the type of identity token that is in the message.

   The type can have one of the following values:

   - `Transport Default` (on the security enabled input nodes)
   - `Current token` (on the SecurityPEP node)
   - `Username`
   - `Username and password`
   - `X.509 Certificate`
   - `SAML assertion`
   - `Kerberos GSS v5 AP_REQ` (on the SecurityPEP node)
   - `LTPA v2 token` (on the SecurityPEP node)
   - `Universal WSSE token` (on the SecurityPEP node)

   On the security enabled input nodes, the default value is *Transport Default*. On the SecurityPEP node, the default value is *Current token*, which means that the token type that exists in the identity mapped or source field in the Properties folder is used.
2. In **Identity Token Location**, specify the location in the message where the identity is specified.

   This string is in the form of an ESQL field reference, XPath expression, or string literal, and must resolve to a token with the type `Username`, `Username and password`, `SAML assertion`, `Kerberos GSS v5 AP_REQ`, `LTPA v2 token`, or `X.509 Certificate`. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.).
3. In **Identity Password Location**, enter the location in the message where the password is specified.

   This string is in the form of an ESQL field reference, XPath expression, or string literal, and must resolve to a string containing a password. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.). This option can be set only if the **Identity Token Type** is set to `Username and password`.
4. In **Identity IssuedBy Location**, specify a string or path expression to show where (in the message) information about the issuer of the identity is held.

   This string is in the form of an ESQL field reference, XPath expression, or string literal, defining where the identity was defined. If you use a string literal, it must be enclosed in single quotes and must not contain a period (.).

   If you leave this property blank on the security enabled input nodes, the transport header value is used (if there is one). For example, for MQ the MQMD.PutApplName value is used. If you leave this property blank on the SecurityPEP node, the WS-Trust request is sent to the STS without the optional Issuer element in the WS-Trust message.
5. (Optional) Ensure that all input nodes share the same information by promoting the properties to the message flow.

What to do next

To enable the extraction of an identity in a security enabled input node or SecurityPEP node, select a security profile that has at least one security operation configured (authentication, mapping, or authorization) or propagation enabled:

1. In the IBM App Connect Enterprise Toolkit, right-click the BAR file, then click **Open with** > **BAR Editor**.
2. Click the **Manage and Configure** tab.
3. Click the flow or node on which you want to set the security profile. The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.
4. In the **Security Profile Name** field, select a security profile.
5. Save the BAR file.

Alternatively, you can set a security profile on the flow or the input node by using the **mqsiapplybaroverride** command. For example:

```
mqsiapplybaroverride -b barFileName -k applicationName -m
flowName#nodeName.securityProfileName=securityProfileName
```

For more information, see command.

Configuring identity authentication and security token validation

You can configure a message flow to perform identity authentication or security token validation by using Lightweight Directory Access Protocol (LDAP), a WS-Trust V1.3 compliant Security Token Service (STS) such as Tivoli Federated Identity Manager (TFIM) Version 6.2, or HTTP requests. Support for TFIM V6.1 is also provided, for compatibility with previous versions of IBM App Connect Enterprise. Alternatively, you can configure a message flow or integration server to authenticate requests against credentials that are stored locally in the integration server's vault.

Before you begin

Check that an appropriate security profile exists, or create a new security profile. See “Creating a security profile” on page 2584.

About this task

For information about configuring authentication, see:

- “Authenticating incoming requests with LDAP” on page 2593
- “Authenticating incoming requests with WS-Trust v1.3 STS (TFIM V6.2)” on page 2595
- “Authenticating incoming requests with TFIM V6.1” on page 2596
- “Authenticating incoming requests by using credentials stored in the vault” on page 2597
- “Configuring authentication for all HTTP and SOAP input nodes in an independent integration server” on page 2598
- “Providing credentials in HTTP requests” on page 2599

Authenticating incoming requests with LDAP

Configure a message flow to perform identity authentication using Lightweight Directory Access Protocol (LDAP).

Before you begin

Before you can configure a message flow to perform identity authentication using LDAP, you need to check that an appropriate security profile exists, or create a new security profile. See “Creating a security profile for LDAP” on page 2584.

About this task

To authenticate the identity of a user or system, the integration server attempts to connect to the LDAP server using the username and password associated with the identity. To do this, the integration server needs the following information:

- To resolve the username to an LDAP entry, the integration server needs to know the base distinguished name (base DN) of the accepted login IDs. This is required to enable the integration server to differentiate between different entries with the same name.
- If the identities do not all have a common base DN, but can be uniquely resolved from a subtree, the DN can be specified in the integration server configuration. When a subtree search has been specified, the integration server must first connect to the LDAP server and search for the given username in order to obtain the full username distinguished name (DN) to be used for authentication. If your LDAP directory does not permit login of unrecognized IDs, and does not grant search access rights on the subtree, you

must set up a separate authorized login ID that the integration server can use for the search. Use the **mqsisetdbparms** command to specify a username and password. For example:

```
mqsisetdbparms -w workDir-n ldap::LDAP -u username -p password
```

or

```
mqsisetdbparms -w workDir -n ldap::<servername> -u username -p password
```

where *<servername>* is your base LDAP server name, for example, `ldap.mydomain.com`.

If you specify `ldap::LDAP`, it creates a default setting for the integration server, which the integration server attempts to use if you have not explicitly used the **mqsisetdbparms** command to create a login ID for a specific *<servername>*. All servers that do not have an explicit `ldap::servername` entry then start using the credentials in the `ldap::LDAP` entry. This means that any servers that were previously using anonymous bind by default will start using the details in `ldap::LDAP`.

The username that you specify in the **-u** parameter must be recognized by the LDAP server as a complete user name. In most cases this means that you need to specify the full DN of the user. Alternatively, by specifying a username to be anonymous, you can force the integration server to bind anonymously to this LDAP server. This might be useful if you have specified a non-anonymous bind as your default (ldap::LDAP). For example:

```
mqsisetdbparms -w workDir -n ldap::<servername> -u anonymous -p password
```

In this case, the value specified for *password* is ignored.

Steps for enabling LDAP authentication:

Procedure

To enable an existing message flow to perform identity authentication, use the BAR editor to select a security profile that uses LDAP for authentication.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) Switch to the Integration Development perspective.

b) In the Application Development view, right-click the BAR file and then click **Open with** > **BAR Editor**.

c) Click the **Manage and Configure** tab.

d) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

e) In the **Security Profile Name** field, select a security profile that uses LDAP for authentication.

f) Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see “Policy sets” on page 2662.

If the message identity does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from IBM MQ with only a username, the password information must be taken from the message body. For more information, see “Configuring the extraction of an identity or security token” on page 2590.

Authenticating incoming requests with WS-Trust v1.3 STS (TFIM V6.2)

You can configure supported message flow input nodes or SecurityPEP nodes to perform identity authentication or security token validation using a WS-Trust v1.3 compliant Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

Before you begin

Before you can configure identity authentication or token validation, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

About this task

When you use a WS-Trust v1.3 STS for authentication, a request is made to the trust service with the following parameters, which control the STS processing. If you are using TFIM V6.2, the following parameters are used in the selection of the TFIM module chain:

- RequestType
- Issuer
- AppliesTo

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574 .

The WS-Trust v1.3 specification, published by OASIS, is available at:

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html
```

.

Steps for enabling WS-Trust v1.3 authentication:

Procedure

To enable an existing message flow to perform authentication or token validation, use the BAR file editor to select a security profile that uses a WS-Trust v1.3 STS for authentication and to associate it with the node or message flow.

If a security profile is specified on either a message flow or a node, the profile must be available when the message flow is deployed; otherwise, a deployment error occurs.

a) In the IBM App Connect Enterprise Toolkit, right-click the BAR file, then click **Open with** > **BAR Editor**.

b) Click the **Manage and Configure** tab.

c) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

d) In the **Security Profile Name** field, select a security profile that configures WS-Trust v1.3 STS for authentication.

e) Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified.

If the message identity (or security token) does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from IBM MQ with only a username, the password information must be taken from the message body. For more information, see "Configuring the extraction of an identity or security token" on page 2590.

Authenticating incoming requests with TFIM V6.1

You can configure a message flow to perform identity authentication by using Tivoli Federated Identity Manager (TFIM) V6.1.

Before you begin

Before you can configure a message flow to perform identity authentication, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for TFIM V6.1" on page 2589.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of IBM App Connect Enterprise. If possible, upgrade to TFIM V6.2 and follow the instructions in "Authenticating incoming requests with WS-Trust v1.3 STS (TFIM V6.2)" on page 2595.

When you use TFIM V6.1 for authentication, a request is made to the TFIM trust service with the following three parameters, which select the module chain:

- Issuer = Properties.IdentitySourceIssuedBy
- Applies To = The Fully Qualified Name of the Flow: <*integrationNodeName*>.<*Integration Server Name*>.<*Message Flow Name*>
- Token = Properties.IdentitySourceToken

For more information about these parameters, see "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 2572.

For further information about how to configure TFIM, see the IBM Tivoli Federated Identity Manager product documentation online.

Steps for enabling TFIM authentication:

Procedure

To enable an existing message flow to perform identity authentication, use the BAR editor to select a security profile that uses TFIM for authentication.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) Switch to the Integration Development perspective.

b) In the Application Development view, right-click the BAR file, then click **Open with** > **BAR Editor**.

c) Click the **Manage and Configure** tab.

d) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

e) In the **Security Profile Name** field, select a security profile that uses TFIM for authentication.

f) Save the BAR file.

What to do next

If the message identity does not contain enough information for authentication, the information must be taken from the message body. For example, if a password is required for authentication but the message came from IBM MQ with only a username, the password information must be taken from the message body. For more information, see "Configuring the extraction of an identity or security token" on page 2590.

Authenticating incoming requests by using credentials stored in the vault

Implement basic authentication on incoming requests by using credentials stored locally in an independent integration server's vault.

About this task

You can secure your integrations by implementing basic authentication on deployed integrations, so that incoming HTTP requests must supply a username and password, which are then authenticated against credentials that are stored in an independent integration server's vault.

Procedure

To configure the authentication of incoming requests by using credentials stored in the vault, complete the following steps:

1. Create a Security Profiles policy in a Policy project (for example, `SecPolicy/BasicAuthLocal.policyxml`) and configure authentication against locally stored credentials, by setting the **Authentication** property to `Local` and the **AuthenticationConfiguration** property to the name of a local credential alias. For example:

```
<policies>
  <policy policyType="SecurityProfiles" policyName="BasicAuthLocal"
 policyTemplate="SecurityProfiles">
    <authentication>Local</authentication>
    <authenticationConfig>LocalCredentialsAlias</authenticationConfig>
    <mapping>NONE</mapping>
    <mappingConfig></mappingConfig>
    <authorization>NONE</authorization>
    <authorizationConfig></authorizationConfig>
    <propagation>false</propagation>
    <idToPropagateToTransport>Message ID</idToPropagateToTransport>
    <transportPropagationConfig></transportPropagationConfig>
    <keyStore>Reserved for future use</keyStore>
    <trustStore>Reserved for future use</trustStore>
    <passwordValue>PLAIN</passwordValue>
    <rejectBlankpassword>false</rejectBlankpassword>
  </policy>
</policies>
```

   For information about how to create a policy, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.

2. Deploy the policy project that contains the new security profile, by using one of the following methods:
   - Add the policy project to a BAR file and deploy it
   - Copy the policy project to the integration server's overrides directory (*work_directory*`/overrides`)
3. Create the vault for the integration server, by using the command; for example:

```
mqsivault --work-dir c:\mywrk\myaceworkdir --create --vault-key abcd1234
```

   For more information about creating a vault, see "Creating a vault by using the mqsivault command" on page 177.

4. Create the local credentials with the configured alias name in the integration server's vault, by using the command; for example:

```
mqsicredentials --work-dir c:\mywrk\myaceworkdir --create --vault-key abcd1234 --credential-
type local
--credential-name LocalCredentialsAlias --username SecUserName --password SecPwd
```

   For more information about creating security credentials, see "Configuring encrypted credentials by using the mqsicredentials command" on page 177.

5. Set the security profile to apply to the whole message flow, to a specific message flow node, or to the whole independent integration server, by completing one of the following steps:
   - Set the security profile to apply to the whole message flow by creating a BAR override, setting the security profile to *{policy_project_name}:security_profile_name* as a property on the message flow
   - Set the security profile to apply to a specific message flow node in either of the following ways:
     - Create a BAR override, setting the security profile to *{policy_project_name}:security_profile_name* as a property on the message flow node
     - Set the **Security profile** property in the **Security** tab of the appropriate message flow node. The **Security profile** property can be set on the following nodes:
       - CICSRequest
       - HTTPInput, HTTPRequest, HTTPAsyncRequest
       - IMSRequest
       - MQInput, MQOutput, MQReply
       - RESTRequest, RESTAsyncRequest
       - SAPRequest
       - SecurityPEP
       - SiebelRequest
       - SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest
   - Set the security profile to apply to the whole integration server, by setting the **forceServerHTTPSecurityProfile** property in the integration server's `server.conf.yaml` configuration file to the name of the policy, in the form *{policy_project_name}:security_profile_name*; for example:

     ```
     forceServerHTTPSecurityProfile: '{SecPolicy}:BasicAuthLocal'
     ```

     The setting specified in the **forceServerHTTPSecurityProfile** property in the `server.conf.yaml` file overrides any specific settings made at the level of the message flow or message flow node.
6. Restart the integration server for the changes to take effect, ensuring that you specify the vault key; for example:

   ```
   IntegrationServer --work-dir c:\mywrk\myaceworkdir --vault-key abcd1234
   ```

What to do next

When the deployed message flows are invoked with security applied, any request to the flow must provide the required credentials. Failure to do so will result in an authentication error and the request will fail.

Configuring authentication for all HTTP and SOAP input nodes in an independent integration server

Enforce authentication on all HTTP and SOAP input nodes deployed to an independent integration server.

About this task

You can configure an independent integration server to enforce basic authentication on all HTTP and SOAP input nodes by setting **forceServer** properties in the `server.conf.yaml` configuration file:

```
#forceServerHTTP:               false  # Set true to override and force unsecured HTTP on all
 HTTP/SOAP input nodes deployed in this server. Defaults to false.
```

```
#forceServerHTTPS:              false  # Set true to override and force secured HTTPS on all
 HTTP/SOAP input nodes deployed in this server. Defaults to false.
#forceServerHTTPSecurityProfile: ''    # Set a security profile, {<policy project
 name>}:<security profile basename>, to override and force all HTTP/SOAP transport
                                       # input nodes deployed in this server to apply the
 security set in the profile. Default is unset, so flow or node setting applies
```

Procedure

To configure the independent integration server to enforce basic authentication on HTTP and SOAP input nodes, complete the following steps:

1. Open the `server.conf.yaml` configuration file for the integration server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. Enforce secured HTTPS on all HTTP and SOAP input nodes deployed to the integration server, by uncommenting the **`forceServerHTTPS`** property and setting it to `true`.

   This setting overrides any existing authentication settings specified on the HTTP and SOAP input nodes and on the message flow.
3. Specify a security profile to apply to all HTTP and SOAP input nodes deployed to the integration server, by setting the **`forceServerHTTPSecurityProfile`** property to the name of the policy, in the form *{policy_project_name}:security_profile_name*; for example:

   ```
   forceServerHTTPSecurityProfile: '{SecPolicy}:BasicAuthLocal'
   ```

   The security settings in the security profile will override any security settings specified on the HTTP and SOAP input nodes and on the message flow. If this property is not set, the settings specified on the input nodes or message flow apply.
4. Restart the integration server for the changes to take effect.

Providing credentials in HTTP requests

Use a security profile to configure HTTPRequest and SOAPRequest nodes to authenticate with a remote server.

About this task

Basic authentication is a common extension in the HTTP protocol that allows a client to provide identity information to a remote web server in the form of a username and password sent in the HTTP header data. Security profiles in IBM App Connect Enterprise provide a way for message flow designers to provide these credentials without building the HTTP headers in a Compute node.

The identity that is to be propagated in the HTTP request can be provided in any of the following ways:

- If identity propagation is enabled for the selected security profile, the HTTPRequest and SOAPRequest nodes automatically pick up username and password credentials, if present, from the Properties tree. This setting is enabled for the predefined Default Propagation security profile. For more information, see “Configuring a message flow for identity propagation” on page 2613.
- You can configure a static username and password identity to be used, by specifying credentials with the command and the command. For information about using these commands to configure credentials, see “Configuring encrypted security credentials” on page 177. Alternatively, you can use the **`mqsisetdbparms`** command. For more information, see “Configuring a message flow for identity propagation” on page 2613.

Note: By default, the credentials are placed in the HTTP request only in response to a 401 challenge response from the server. The response includes a list of authentication mechanisms that are supported by the remote server that is providing the requested service. The integration server selects the highest level authentication protocol from this list of authentication mechanisms based on the set of protocols

that it supports. Nodes then use this protocol for the connection. You can configure the supported protocol by using the **allowedAuthTypes** property of the ComIbmSocketConnectionManager object. You can configure the nodes to send pre-emptively by setting the **preemptiveAuthType** property. For examples of how to set these properties, see "Providing credentials for outbound requests by using IWA" on page 2600.

To enable basic authentication, select an appropriate security profile for the output node or the message flow in the BAR file editor. The credentials are picked up from the following Properties tree locations if set:

```
Properties.IdentityMappedType
Properties.IdentityMappedToken
Properties.IdentityMappedPassword
```

If the mapped identity fields are not set, the credentials are picked up from the following Properties tree locations:

```
Properties.IdentitySourceType
Properties.IdentitySourceToken
Properties.IdentitySourcePassword
```

For basic authentication both a username and password are required, therefore the appropriate Type field must be set to the string `usernameAndPassword`. For example:

```
SET OutputRoot.Properties.IdentitySourceType='usernameAndPassword';
SET OutputRoot.Properties.IdentitySourceToken = 'myUser';
SET OutputRoot.Properties.IdentitySourcePassword = 'myPassw0rd';
SET OutputRoot.Properties.IdentitySourceIssuedBy = 'myDomain';
```

These fields are interpreted by a subsequent HTTPRequest or SOAPRequest node and converted into a basic authentication HTTP header.

You can also propagate credentials from an input message by setting a security profile which includes propagation on an input node, and then using the input node properties `Identity token type`, `Identity Token location` and `Identity password location`. These three properties take an XPath expression that specifies the location in the input message to retrieve the appropriate token or password from. When configured correctly, these properties place the identity information in the `Properties.IdentitySourceType`, `Properties.IdentitySourceToken` and `Properties.IdentitySourcePassword` fields. HTTPRequest or SOAPRequest nodes then use these values directly, with an appropriate security policy.

You can override the configuration of the security profile by selecting the build option **Override configurable property values** in the BAR file editor.

Providing credentials for outbound requests by using IWA
Set up IBM App Connect Enterprise to consume a remote service that is secured with Integrated Windows Authentication (IWA). Only IBM App Connect Enterprise running on Windows can consume an IWA-secured service.

Before you begin

Your IBM App Connect Enterprise must be running on the Windows operating system. If it is running on a different operating system, an IWA-secured remote service cannot be consumed.

Your message flow must include one or more of the following nodes:

- HTTPRequest
- SOAPRequest
- RESTRequest

You cannot use the HTTPAsyncRequest, SOAPAsyncRequest, or RESTAsyncRequest nodes to consume a remote service that is secured with Integrated Windows Authentication (IWA). If your message flow includes an HTTPRequest node, you must set the **HTTP version** property to *1.1* and select *Enable HTTP/1.1 keep-alive* on the **HTTP Settings** tab in the **Properties** view of the node.

A security identity is required for outbound authentication. By default, the identity credentials of the integration node user ID (the **serviceUserId** parameter that is specified by the **mqsicreatebroker** command) is sent to the remote service to use for authentication. If you require a specific security identity to be propagated, you must set the appropriate identity credentials in the Properties tree. For more information, see "Providing credentials in HTTP requests" on page 2599.

About this task

Use the following commands to set up and manage outbound support for the NTLM, Kerberos, SPNEGO, and SPNEGO-2 protocols, which together are referred to as Integrated Windows Authentication (IWA). By default IWA is enabled.

To consume a remote service that is secured with IWA, stop the integration node and run the following command:

```
mqsichangeproperties integrationNodeName -e integrationServerName -o
 ComIbmSocketConnectionManager
 -n allowedAuthTypes -v "PropertyValue" -f
```

Where:

- *integrationNodeName* is the name of the integration node you want to modify.
- *integrationServerName* is the name of the integration server on that integration node.
- *PropertyValue* is one of the following values:

  IWA
  Allow the integration node to authenticate by using any IWA protocol.

  NTLM
  Allow the integration node to authenticate by using the NTLM protocol.

  Negotiate
  Allow the integration node to authenticate by using the SPNEGO process to negotiate the use of the NTLM or Kerberos protocols.

  Nego2
  Allow the integration node to authenticate by using the SPNEGO-2 process to negotiate the use of the NTLM or Kerberos protocols.

  Basic
  Allow authentication with Basic Authentication.

  All
  Allow authentication with any supported protocol from this list.

  None
  Do not authenticate.

  Multiple values can be given, separated by a semicolon or a space, and these values are not case-sensitive. IBM App Connect Enterprise selects one value from the list of supported IWA protocols by the server, in the following order: Nego2, Negotiate, NTLM.

When security is enabled, the HTTPRequest and SOAPRequest nodes wait for a 401 response from the server that indicates which authentication mechanisms the server supports. The nodes then use the highest supported protocol for the connection, which is selected in the order that is listed previously. When connected, this protocol is used to authenticate pre-emptively until the flow is stopped or the allowedAuthTypes property is changed. To configure any of the protocols to be used pre-emptively, stop the integration node and run the following command:

```
mqsichangeproperties integrationNodeName -e integrationServerName -o
 ComIbmSocketConnectionManager
 -n preemptiveAuthType -v "PropertyValue" -f
```

Where:

- *integrationNodeName* is the name of the integration node you want to modify.

- *integrationServerName* is the name of the integration server on that integration node.
- *PropertyValue* is one of the following values:

  Basic
  Pre-emptively authenticate by using Basic Authentication.

  NTLM
  Pre-emptively authenticate by using the NTLM protocol.

  Negotiate
  Pre-emptively authenticate by using the SPNEGO process to negotiate the use of the NTLM or Kerberos protocols.

  Nego2
  Pre-emptively authenticate by using the SPNEGO-2 process to negotiate the use of the NTLM or Kerberos protocols.

For more advanced scenarios, the following optional configuration properties can also be used with the ComIbmSocketConnectionManager object:

allowNtlmNegotiation='*TRUE*'
Set to '*FALSE*' to prevent NTLM from being negotiated with the SPNEGO and SPNEGO-2 protocols. The default value is '*TRUE*'.

negotiateMutualAuth='*FALSE*'
Set to '*TRUE*' if you require mutual authentication when the Kerberos protocol is negotiated. The default value is '*FALSE*'.

Note: When IBM App Connect Enterprise is authenticating by using Kerberos, the integration node automatically generates a service principal name (SPN) for the service that is based on the host name for the request. For example, if the URL for the service is http://iib.iibservice/testservice/service1.svc the SPN is assumed to be HTTP/`iib.iibservice`. If the service exists at a different SPN, use the following local environment overrides to provide an explicit SPN for the service:

HTTP
```
SET OutputLocalEnvironment.Destination.HTTP.ServicePrincipalName = 'HTTP/
iib.iibservice2.com:7800';
```

REST
```
SET OutputLocalEnvironment.Destination.REST.Request.ServicePrincipalName =
'HTTP/iib.iibservice2.com:7800';
```

SOAP
```
SET
OutputLocalEnvironment.Destination.SOAP.Request.Transport.HTTP.ServicePrincipalName
= 'HTTP/iib.iibservice2.com:7800';
```

To check the current outbound authentication setting, start the integration node and run the following command:

```
mqsireportproperties integrationNodeName -e integrationServerName
 -o ComIbmSocketConnectionManager -r
```

The following output is displayed within the connector properties:

- allowedAuthTypes='*PropertyValue*'

Where *PropertyValue* is *NTLM*, *Negotiate*, *Nego2*, *None*, or *Basic*. If multiple values are set, they are separated by a semicolon.

Examples

To enable all IWA protocols, stop the integration node and run the following command:

```
mqsichangeproperties INODE -e default -o ComIbmSocketConnectionManager
 -n allowedAuthTypes -v "IWA" -f
```

To enable NTLM and Negotiate (SPNEGO) protocols, stop the integration node and run the following command:

```
mqsichangeproperties INODE -e default -o ComIbmSocketConnectionManager
 -n allowedAuthTypes -v "NTLM;Negotiate" -f
```

To disable all outbound security protocols, stop the integration node and run the following command:

```
mqsichangeproperties INODE -e default -o ComIbmSocketConnectionManager
 -n allowedAuthTypes -v "None" -f
```

Configuring identity mapping

Configure a Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile. For information about security profiles, see “Creating a security profile” on page 2584.

About this task

IBM App Connect Enterprise provides support for identity mapping (also known as identity federation) and token issuance and exchange. Identity mapping is the process of mapping an identity in one realm to another identity in a different realm. For example, you might map *User001* from the eSellers realm to *eSellerUser01* in the eShipping realm. Token issuance and exchange involves the mapping of a token of one type to a token of a different type. For example, an incoming Username and Password token from a client over MQ might be mapped into an equivalent SAML assertion, to be propagated to a web services call. Alternatively, you might exchange a SAML 1.1 assertion from a client application for an equivalent SAML 2.0 assertion for an updated backend server.

For information about configuring identity mapping with either a WS-Trust V1.3 STS (for example, TFIM V6.2) or with TFIM V6.1, see:

- “Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2)” on page 2603
- “Configuring identity mapping with TFIM V6.1” on page 2604

Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2)

Configure a WS-Trust V1.3 compliant Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2, to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile. For information about security profiles, see “Creating a security profile” on page 2584.

About this task

To configure TFIM V6.2 to map an incoming security token, you must create a custom module chain in TFIM, which performs the security operations. The TFIM configuration controls the token type that is returned from the mapping.

When you use a WS-Trust V1.3 STS for identity mapping, a request is made to the security token server with the following parameters, which control the STS processing:

- RequestType

- Issuer
- AppliesTo

If you are using TFIM V6.2, these parameters are used in the selection of the module chain.

The security manager invokes the WS-Trust v1.3 provider only once, even if it is set for additional security operations (such as authentication or authorization). As a result, when you are using TFIM V6.2, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

When the security profile includes a mapping operation, the STS (for example, TFIM V6.2) must return a security token in its response. The STS can return the original unmodified token if no token exchange is required.

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574 .

The WS-Trust v1.3 specification, published by OASIS, is available at:

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html
```

For information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager product documentation online.

Follow these steps to enable an existing message flow to perform identity mapping.

Procedure

Using the BAR editor, select a security profile that has mapping enabled.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) In the IBM App Connect Enterprise Toolkit, right-click the BAR file, then click **Open with** > **BAR Editor**.

b) Click the **Manage and Configure** tab.

c) Click the message flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

d) In the **Security Profile Name** field, enter the name of a security profile that has mapping enabled.

e) Save the BAR file.

Configuring identity mapping with TFIM V6.1

Configure Tivoli Federated Identity Manager (TFIM) V6.1 to map the incoming security token and, if required, to authenticate and authorize it.

Before you begin

Before you can configure a message flow to perform identity mapping, you need to check that an appropriate security profile exists, or create a new security profile. For information about security profiles, see "Creating a security profile" on page 2584.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of IBM App Connect Enterprise. If possible, upgrade to TFIM V6.2 and follow the instructions in "Configuring identity mapping with a WS-Trust V1.3 STS (TFIM V6.2)" on page 2603.

To configure TFIM V6.1 to map the incoming security token, you need to create a custom module chain in TFIM, which performs the security operations. The TFIM configuration controls the token type that is returned from the mapping.

When you use TFIM for mapping, a request is made to the TFIM trust service with the following three parameters, which select the module chain:

- Issuer = Properties.IdentitySourceIssuedBy
- AppliesTo = The fully qualified name of the flow: *integrationNodeName.Integration Server Name.Message Flow Name*
- Token = Properties.IdentitySourceToken

The security manager invokes the security provider only once, even if it is set for additional security operations (such as authentication or authorization). As a result, when you are using TFIM V6.1, you must configure a single module chain to perform all the required authentication, mapping, and authorization operations.

For information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager product documentation online.

Follow these steps to enable an existing message flow to perform identity mapping.

Procedure

Using the BAR editor, select a security profile that has mapping enabled.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) In the IBM App Connect Enterprise Toolkit, right-click the BAR file, then click **Open with** > **BAR Editor**.

b) Click the **Manage and Configure** tab.

c) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

d) In the **Security Profile Name** field, enter the name of a security profile that has mapping enabled.

e) Save the BAR file.

Configuring authorization

Configure the integration node to use an external security provider to authorize an identity in a message flow. You can use either Lightweight Directory Access Protocol (LDAP) or a WS-Trust V1.3 compliant Security Token Service (STS) such as Tivoli Federated Identity Manager (TFIM) V6.2. Support for TFIM V6.1 is also provided for compatibility with previous versions.

Before you begin

Check that an appropriate security profile exists, or create a new security profile. See “Creating a security profile” on page 2584.

About this task

For information about configuring authorization for LDAP, WS-Trust V1.3 (TFIM V6.2), or TFIM V6.1, see:

- “Configuring authorization with LDAP” on page 2606
- “Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)” on page 2607
- “Configuring authorization with TFIM V6.1” on page 2610

What to do next

Configuring authorization with LDAP

This topic describes how to configure a message flow to perform authorization on an identity using Lightweight Directory Access Protocol (LDAP).

Before you begin

Before you can configure a message flow to perform authorization, you need to check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for LDAP" on page 2584.

About this task

When LDAP is used for authorization, the integration server needs to determine whether the incoming username is a member of the given group. To do this, the integration server requires the following information:

- To resolve the username to an LDAP entry, the integration server needs to know the base distinguished name (Base DN) of the accepted login IDs. This is required to enable the integration server to differentiate between different entries with the same name.
- To get an entry list from a group name, the group name must be the distinguished name of the group, not just a common name. An LDAP search is made for the group, and the username is checked by finding an entry matching the distinguished name of the user.
- If your LDAP directory does not permit login by unrecognized IDs, and does not grant search access rights on the subtree, you must set up a separate authorized login ID that the integration server can use for the search. Use the **mqsisetdbparms** command to specify a username and password:

  ```
  mqsisetdbparms -w workDir -n ldap::LDAP -u username -p password
  ```

  or

  ```
  mqsisetdbparms -w workDir -n ldap::<servername> -u username -p password
  ```

  where *<servername>* is your base LDAP server name. For example: `ldap.mydomain.com`.

  If you specify `ldap::LDAP`, it creates a default setting for the integration server, which the integration server attempts to use if you have not explicitly used the **mqsisetdbparms** command to create a login ID for a specific *<servername>*. All servers that do not have an explicit `ldap::servername` entry then start using the credentials in the `ldap::LDAP` entry. This means that any servers that were previously using anonymous bind by default will start using the details in `ldap::LDAP`.

  The username that you specify in the **-u** parameter must be recognized by the LDAP server as a complete user name. In most cases this means that you need to specify the full DN of the user. Alternatively, by specifying a username to be anonymous, you can force the integration server to bind anonymously to this LDAP server. This might be useful if you have specified a non-anonymous bind as your default (ldap::LDAP). For example:

  ```
  mqsisetdbparms -w workDir -n ldap::<servername> -u anonymous -p password
  ```

  In this case, the value specified for *password* is ignored.

Steps for enabling LDAP authorization:

Procedure

To enable an existing message flow to perform authorization using LDAP, use the BAR editor to select a security profile that has authorization enabled.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) Switch to the Integration Development perspective.

b) In the Application Development view, right-click the BAR file and then click **Open with** > **BAR Editor**.

c) Click the **Manage and Configure** tab.

d) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

e) In the **Security Profile Name** field, select a security profile that uses LDAP for authorization.

f) Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 2662.

Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)

You can configure supported message flow input nodes or SecurityPEP nodes to perform authorization of an identity or security token by using a WS-Trust v1.3 compliant Security Token Service (STS), such as Tivoli Federated Identity Manager (TFIM) V6.2.

Before you begin

Before you configure a message flow to perform authorization with a WS-Trust v1.3 STS:

- Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

About this task

The message flow security manager issues an authorization request to the WS-Trust service with the following parameters, which select the TFIM module chain to be used:

- RequestType
- Issuer
- AppliesTo

For more information about these parameters, see:"Authentication, mapping, and authorization with TFIM V6.2 and TAM" on page 2574 .

In addition to configuring IBM App Connect Enterprise to perform authorization with a WS-Trust compliant STS, you must configure TAM. For information about how to do this, see the following topics:

- "Creating a module chain in TFIM V6.2" on page 2608
- "Configuring TAM for authorization using TFIM V6.2" on page 2608

Steps for enabling authorization using a WS-Trust v1.3 STS provider:

Procedure

To enable an existing message flow to enforce authorization using a WS-Trust v1.3 STS provider, use the BAR editor to select a security profile that has authorization set for that provider.

You can set a security profile on a message flow or on individual input nodes or SecurityPEP nodes. If you leave the **`Security Profile`** property blank, the node inherits the **`Security Profile`** property that is set at the message flow level. If you leave the property blank at both levels, security is turned off for the node.

a) In the IBM App Connect Enterprise Toolkit, right-click the BAR file, then click **Open with** > **BAR Editor**.

b) Click the **Manage and Configure** tab.

c) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

d) In the **Security Profile Name** field, select a security profile that has authorization set for *WS-Trust V1.3 STS*.

e) Save the BAR file.

What to do next

For a SOAPInput node to use the token in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified.

The WS-Trust v1.3 specification is available at: `http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html`.

Creating a module chain in TFIM V6.2
This topic describes how to create a module chain in Tivoli Federated Identity Manager (TFIM) V6.2.

About this task

When you use a WS-Trust v1.3 Security Token Service (STS) for authentication, authorization, or mapping (or any combination of those operations), a single WS-Trust request is made to the trust service with the required parameters, which control the STS processing.

To enable IBM App Connect Enterprise to use TFIM V6.2 for authorization, you need to configure TFIM to process the single WS-Trust request from the integration node security manager. To configure TFIM, you must create a module chain to handle the request:

Procedure

1. Create a *Custom* module chain, and ensure that the chain performs all the actions that are specified in the integration node security profile (Authenticate, Map, Authorize).
2. Set the *RequestType*, *Issuer* and *AppliesTo* properties of the module chain, so that it is invoked for the requests from the security enabled input node or SecurityPEP node.

   The parameters that are passed by the integration node to TFIM are shown in the table in “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 2574.

What to do next

If your module chain includes an authorization module, and if the module specifies TAM, you must configure TAM to process the authorization requests from TFIM. For more information about how to do this, see “Configuring TAM for authorization using TFIM V6.2” on page 2608.

Configuring TAM for authorization using TFIM V6.2
Configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.2.

About this task

To configure TAM for a TFIM V6.2 TAMAuthorizationSTSModule, complete the following steps using the pdadmin utility.

Procedure

1. Check that the **`action group`** used by the TFIM authorization module is available.

   The action group used is *WebService*:

```
action group list
```

If *WebService* is not listed, create it:

```
action group create WebService
```

2. Display the **action** in the action group used by the TFIM authorization module.
   The action used is *"i"*:

```
action list WebService
```

   If action *"i" <label> 0* is not listed, create it. The value of *<label>* can vary:

```
action create i <label> 0 WebService
```

3. Create the Access Control List (ACL) that will be used to grant access to one or more message flows.
   First, create the ACL and give the administrators access to it. In this example, `iv-admin` is the administration group and `sec_master` is the main administrator:

```
acl create <AclName>
acl modify <AclName> set Group iv-admin TcmdbsvaBRxl[WebService]i
acl modify <AclName> set User sec_master TcmdbsvaBRxl[WebService]i
```

4. Grant access to all authenticated users, or specific groups, by adding them to the ACL. Grant any authenticated identity access:

```
acl modify <AclName> set Any-other Trx[WebService]i
```

   To add a specific group:

```
acl modify <AclName> set group <GroupName> Trx[WebService]i
```

   In these strings, each occurrence of `Trx[ ]` is an action, and corresponds to the value of the stsuser Action context attribute that is passed into the TAMAuthorizationSTSModule. For more information, see “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 2574.

5. Create a protected object space path in TAM to correspond to the value of the stsuser ObjectName context attribute that is passed into the TAMAuthorizationSTSModule using the following command syntax:

```
objectspace create /<ObjectName>
```

   For more information, see “Authentication, mapping, and authorization with TFIM V6.2 and TAM” on page 2574.

6. Attach the ACL to the protected object space path that you have created.
   Each node in the object space inherits ACLs from its parent, and a lower level ACL can override a higher level one. Use the following command syntax to attach an ACL to a node in the object space path:

```
acl attach /<ObjectSpacePath> <AclName>
```

Configuring authorization with TFIM V6.1

You can configure a message flow to perform authorization on an identity by using Tivoli Federated Identity Manager (TFIM) V6.1.

Before you begin

Before you configure a message flow to perform authorization with TFIM V6.1:

- Check that an appropriate security profile exists, or create a new security profile. See "Creating a security profile for TFIM V6.1" on page 2589.
- Define the required users and groups in TFIM.

About this task

Note: Support for TFIM V6.1 is included for compatibility with previous versions of IBM App Connect Enterprise. If possible, upgrade to TFIM V6.2 and follow the instructions in "Configuring authorization with a WS-Trust v1.3 STS (TFIM V6.2)" on page 2607.

The integration node security manager issues an authorization request to the TFIM trust service with the following three parameters, which select the TFIM module chain to be used:

- Issuer = Properties.IdentitySourceIssuedBy
- Applies To = The Fully Qualified Name of the Flow: <*integrationNodeName*>.<*Integration Server Name*>.<*Message Flow Name*>
- Token = Properties.IdentitySourceToken

Authorization is performed with TFIM using an instance of the TFIM AuthorizationSTSModule in the selected module chain. The TFIM AuthorizationSTSModule must be set with `Mode = Other`. This AuthorizationSTSModule authorizes a user by checking an Access Control List (ACL) from Tivoli Access Manager (TAM). TFIM performs the authorization check by verifying that the action "i" (invoke) has been granted in an ACL for the `WebService` action group.

The ACL is found starting from the root of the TAM object space using a path formed from the Authorization module **`Web service protected object name`** parameter, followed by the **`Port Type`** and **`Operation Name`** from the authorization request. When the integration node makes an authorization request to TFIM, the **`Port Type`** and **`Operation Name`** parameters have the following values:

- PortType:<*Message flow name*>
- Operation "MessageFlowAccess"

Therefore, the ACL is found at this location in the TAM object space:

```
/<WSProtectedObjectName>.<MessageFlowName>."MessageFlowAccess"
```

For more information about this process and the parameters, see "Authentication, mapping, and authorization with TFIM V6.1 and TAM" on page 2572.

Steps for enabling TFIM authorization:

Procedure

To enable an existing message flow to perform authorization with TFIM, use the BAR editor to select a security profile that has authorization enabled.

You can set a security profile on a message flow or on individual input nodes. If no security profile is set for the input nodes, the setting is inherited from the setting on the message flow.

a) Switch to the Integration Development perspective.

b) In the Application Development view, right-click the BAR file, then click **Open with** > **BAR Editor**.

c) Click the **Manage and Configure** tab.

d) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

e) In the **Security Profile Name** field, select a security profile that has authorization enabled.

f) Save the BAR file.

What to do next

For a SOAPInput node to use the identity in the WS-Security header (rather than an underlying transport identity) an appropriate policy set and bindings must also be defined and specified. For more information, see "Policy sets" on page 2662.

In addition to configuring IBM App Connect Enterprise to perform authorization with TFIM, you must configure TFIM and TAM. For information about how to do this, see the following topics:

- "Creating a module chain in TFIM V6.1" on page 2611
- "Configuring TAM for authorization using TFIM V6.1" on page 2612.

For further information on how to configure TFIM, see the IBM Tivoli Federated Identity Manager product documentation online.

Creating a module chain in TFIM V6.1

This topic describes how to create a module chain in Tivoli Federated Identity Manager (TFIM) V6.1.

About this task

To enable IBM App Connect Enterprise to use TFIM V6.1 for authorization, you need to configure TFIM to process the security request from the message flow. To do this you need to create a module chain in TFIM to handle the request:

Procedure

1. Create a *Custom* module chain, and ensure that the chain performs all the actions required (Authenticate, Map, Authorize).
2. Set the *Issuer* and *AppliesTo* properties of the module chain, so that it is invoked for the requests from the message flow.

   When the integration node makes a request to TFIM, the **`Port Type`** and **`Operation Name`** parameters have the following values:

   - PortType:*<Message flow name>*
   - Operation "MessageFlowAccess"

   The *RequestType* is always set to *Validate*.
3. To perform authorization in a module chain, add an instance of the Authorization module in *other* mode, which allows the module parameter **`Web Service protected object name`** to be set for the Tivoli Access Manager (TAM) configuration.

What to do next

When you have created the module chain in TFIM, see "Configuring TAM for authorization using TFIM V6.1" on page 2612 for information on how to configure TAM to process authorization requests from TFIM.

Configuring TAM for authorization using TFIM V6.1
This topic describes how to configure Tivoli Access Manager (TAM) to enable authorization using Tivoli Federated Identity Manager (TFIM) V6.1.

About this task

To configure TAM to process an authorization request from TFIM, complete the following steps. The examples relate to the TAM Version 6.01 pdadmin utility:

Procedure

1. Check that the **action group** used by the TFIM authorization module is available.
   The action group used is *WebService*:

   ```
   action group list
   ```

   If *WebService* is not listed, create it:

   ```
   action group create WebService
   ```

2. Display the **action** in the action group used by the TFIM authorization module.
   The action used is *"i"*:

   ```
   action list WebService
   ```

   If action *"i" <label> 0* is not listed, create it. The value of *<label>* can vary:

   ```
   action create i <label> 0 WebService
   ```

3. Create the Access Control List (ACL) that will be used to grant access to one or more message flows.
   First, create the ACL and give the administrators access to it. In this example, `iv-admin` is the administration group and `sec_master` is the main administrator:

   ```
   acl create <AclName>
   acl modify <AclName> set Group iv-admin TcmdbsvaBRxl[WebService]i
   acl modify <AclName> set User sec_master TcmdbsvaBRxl[WebService]i
   ```

4. Grant access to all authenticated users, or specific groups, by adding them to the ACL. Grant any authenticated identity access:

   ```
   acl modify <AclName> set Any-other Trx[WebService]i
   ```

   To add a specific group:

   ```
   acl modify <AclName> set group <GroupName> Trx[WebService]i
   ```

5. Define protected object spaces in TAM for authorization of message flows:
   a) Create the *application container object* as the root of the protected object space.
      This is the name that is used to link an instance of a TFIM AuthorizationSTSModule (within a module chain) into the TAM object space. The container object name is specified to match the **Web Service protected object name** parameter on a TFIM Authorization module.

```
objectspace create /<ContainerObjectName> <Description> 14
```

b) Create the container objects in the tree for each integration node message flow that is being authorized.

The message flow name is used by TFIM to locate a point in the TAM Object Space tree for Authorization, through the attached ACL. The message flow name is passed as the **PortType** in the WS-Trust request to TFIM. Use the following command to create the object tree node representing each flow to be authorized:

```
object create /<ContainerObjectName>/<FlowName> <Description> 11 ispolicyattachable yes
```

The **ispolicyattachable** parameter applies to all levels, so you can attach an ACL at any level.

c) Create the leaf object that represents the authorized object to grant access to the message flow.

This is the fixed string *MessageFlowAccess*, which the integration node sends to TFIM through the TFIM **OperationName** extension to the WS-Trust request. A fixed name (MessageFlowAccess) is used instead of a true operation name, because the integration node does not necessarily know at the input node which operation a flow is going to perform. The command syntax is:

```
object create /<ContainerObjectName>/<FlowName>/MessageFlowAccess <Description> 12 ispolicyattachable
 yes
```

where *<FlowName>* has been created in a previous step.

6. Attach the ACL to the relevant node in the protected object space tree.

Each node in the object space inherits ACLs from its parent, and a lower level ACL can override a higher level one. Use the following command syntax to attach an ACL to a node in the object space:

```
acl attach /<ObjectSpacePath> <AclName>
```

To attach an ACL to the leaf node:

```
acl attach /<ContainerObjectName>/<FlowName>/MessageFlowAccess <AclName>
```

Configuring a message flow for identity propagation

To enable a message flow to perform identity propagation, the input nodes must extract the identity from the message flow and the output node must propagate it. If the message identity does not contain enough information for identity propagation, you can provide the identity to propagate.

Before you begin

Before you can configure a message flow to perform identity propagation, you must check that an appropriate security profile exists, or create a new security profile. See “Creating a security profile” on page 2584.

About this task

To configure a message flow to perform identity propagation, complete the following tasks:

1. “Enabling identity propagation” on page 2614
2. “Providing the identity to propagate” on page 2614

Enabling identity propagation

About this task

An input node extracts security tokens if it is configured with a security profile at deployment time. An output node propagates an identity if it is configured with a security profile that enables propagation at deployment time.

To enable a message flow to perform identity propagation, complete the following steps.

Procedure

By using the BAR editor, select a security profile that has identity propagation enabled.

You can use the `Default Propagation` profile, which is a predefined profile that requests only identity propagation. You can set a security profile on a message flow or on individual input and output nodes. If no security profile is set for the input and output nodes, the setting is inherited from the setting on the message flow.

a) In the Application Development view, right-click the BAR file, then click **Open with** > **BAR Editor**.

b) Click the **Manage and Configure** tab.

c) Click the flow or node on which you want to set the security profile.

The properties that you can configure for the message flow or for the node are displayed in the **Properties** view.

d) In the **Security Profile Name** field, select a security profile that has identity propagation enabled.

e) Save the BAR file.

Alternatively, you can set a security profile on the flow or the input node by using the **mqsiapplybaroverride** command. For example:

```
mqsiapplybaroverride -b barFileName -k applicationName -m
flowName#nodeName.securityProfileName=securityProfileName
```

For more information, see command.

Providing the identity to propagate

About this task

For information about the identity tokens that you can propagate with each node type, see “Identity and security token propagation” on page 2580.

If the message identity does not contain enough information for identity propagation, you can use any of the following methods to acquire the necessary information:

- Use information that is in the message body. For example, if the message comes from IBM MQ with only a `Username` token, and the output is an HTTPRequest node that requires a `Username and password` token, the password might be present in the body of the incoming message. For more information, see “Configuring the extraction of an identity or security token” on page 2590.
- Configure an identity mapper by using IBM Tivoli Federated Identity Manager. For more information, see the IBM Tivoli Federated Identity Manager product documentation online.
- Use ESQL or Java to set the Mapped Identity fields in the Properties tree.

- Configure a static user name and password identity by completing the following steps:
  1. Run the **mqsisetdbparms** command:

     ```
     mqsisetdbparms -w workDir -n securityIDName -u username -p password
     ```

     Where *securityIDName* is a name to associate with the static user name and password identity, and *username* and *password* are the identity credentials that you want to use. For more information, see command.
  2. Create a Security Profiles policy that sets the property values listed in the following table:

Properties	Values
Propagation	True
Identifier to propagate	Static ID
Transport propagation configuration	*securityIDName*

     Where *securityIDName* is the name that you associated with the static user name and password identity in the **mqsisetdbparms** command.

Securing message flows by using SSL

You can secure message flows by enabling nodes within message flows to use SSL and certificates.

After you establish a public key infrastructure (PKI) configuration for your integrations servers or integration nodes, you can use the PKI configuration within your message flows. For information on establishing a public key infrastructure, see “Setting up a public key infrastructure” on page 2635.

When you configure message flow nodes that make or accept TCP connections, you can set SSL encryption and certificates on those connections.

For more information about using SSL and certificates to secure your message flows, see:

- “Configuring JMS nodes to use to use SSL” on page 2615
- Configuring SOAPRequest and SOAPAsyncRequest nodes to use SSL (HTTPS)
- Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)
- Securing the connection to CICS Transaction Server for z/OS by using SSL
- SSL and the TCP/IP nodes

Configuring JMS nodes to use to use SSL

Configure your JMS nodes to work with a JMS provider that supports JMS clients that can connect by using the Secure Sockets Layer (SSL) protocol.

About this task

The JMS 1.1 Specification states that JMS does not provide features for controlling or configuring message integrity or message privacy. JMS providers typically support these additional features, and provide their own administration tools to configure these services. Clients can get the appropriate security configuration as part of the administered objects that they use.

If you want to apply SSL security to the JMS connections created by the three built-in nodes JMSInput, JMSOutput, and JMSReply, check the documentation supplied by your chosen JMS provider. The configuration of the JNDI administered objects that are used by the JMS nodes is specific to each JMS provider.

The three built-in nodes JMSInput, JMSOutput, and JMSReply are referred to in this topic by the generic term JMS nodes; apply the information and instructions here to the specific type of node that you are using.

One example of a JMS provider that provides SSL support for connecting JMS clients is TIBCO Enterprise Message Service (EMS). The following sections describe the authentication model used for JMS nodes, with specific reference to TIBCO EMS, and provide information about how to connect JMS nodes to a TIBCO EMS JMS Server securely by using SSL:

1. "SSL authentication model for the JMS nodes" on page 2616
2. "Configuring your JMS nodes to use SSL-enabled JNDI administered objects" on page 2616

SSL authentication model for the JMS nodes

About this task

The JMS provider TIBCO EMS supports Java clients that can use either the Java Secure Sockets Extension (JSSE) Java package, or an SSL implementation supplied by Entrust. For details about the services provided, see the documentation provided with your chosen package.

TIBCO EMS supports a number of different authentication scenarios, but JMS nodes can use only client authentication to the server. In this scenario, the TIBCO EMS server requests the client's digital certificate during an SSL handshake, and checks its issuer against the server's list of trusted Certificate Authorities. If the authority is not in the server's list, further communications are prevented with the JMS node.

Therefore, you must configure the EMS server to explicitly enable client authentication of the SSL certificates in its configuration file; configure the JNDI administered SSL JMS connection factories for the same level of support.

Configuring your JMS nodes to use SSL-enabled JNDI administered objects

About this task

The JMS nodes use JNDI to look up a connection factory object that is used to create JMS connections to a TIBCO EMS server.

Procedure

1. Configure the JMS node property Connection factory name to specify a pre-configured connection factory that is enabled for SSL connectivity.

   Make sure that you have set the appropriate parameters in the corresponding SSL JMS connection factory definition:

   - Enable client authentication
   - Specify the SSL protocol in the server URL
   - Set other parameters to define the support your require.

   See the provider's documentation for information about how to generate this JNDI administered object:
2. Configure the JMS node property Location JNDI Bindings with the URL that points to the JNDI bindings containing the JNDI administered objects for SSL connectivity.

   For TIBCO EMS , this URL takes the following format:

   ```
   tibjmsnaming://server_name:ssl_port
   ```

   - *server_name* is the host name of the computer where the server is installed.
   - *ssl_port* is the server port for SSL connectivity; typically, this is port 7243 for a TIBCO EMS server.

3. Make the TIBCO EMS client JAR files available to the integration server to which you deploy the message flow that includes your JMS nodes.

   Set the JMS Providers policy property `Type 4 driver class JARs URL` to point to the directory that contains the JMS provider's client JAR files and the SSL vendor's JAR files.

   Some JMS providers use JSSE to implement SSL support; ensure that the following JAR files are in the `jarsURL` directory:

   - `jsse.jar`
   - `net.jar`
   - `jcert.jar`
   - `tibcrypt.jar`

   You can find standard non-SSL client JAR files in the same location.

   Refer to your JMS provider's documentation for more details on the specific JAR files that are required.

Configuring SOAPInput and SOAPReply nodes to use SSL (HTTPS)

Configure the SOAP nodes to communicate with other applications that use HTTPS by creating a keystore file, and configuring the integration server or integration node to use SSL.

Before you begin

Set up a public key infrastructure (PKI) at integration server level: "Setting up a public key infrastructure" on page 2635.

About this task

Follow these steps to configure the SOAPInput and SOAPReply nodes to communicate with other applications using HTTP over SSL:

1. If you are using the integration node listener: Configure the integration node to use SSL
2. If you are using the integration server (embedded) listener: Configure an integration server to use SSL

If you configured your integration node and integration servers such that the integration node listener is used for some integration servers, and the integration server listener for other integration servers, you must complete step 1 for the first set of integration servers and step 2 for each integration server in the second set.

For information about which listener to use for HTTP messages, see "HTTP listeners" on page 795.

Configuring an integration node to use SSL

If you want to use the integration node listener for HTTPS, configure values for the node's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: If you do not want to use the default port 7083 for HTTPS messages, specify the **port** on which the integration node listens:

```
mqsichangeproperties integrationNodeName
  -b httplistener -o HTTPSConnector
```

```
  -n port -v Port_to_listen_on_for_https
```

On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024.

For the integration node to listen on these ports, the user ID under which the integration node is started must be root.

2. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties integrationNodeName -b httplistener -o HTTPSConnector
  -n ReqClientAuth -v true
```

3. Restart the integration node after you change one or more of the HTTP listener properties.
4. Optional: Use the following commands to display HTTP listener properties:

```
mqsireportproperties integrationNodeName -b httplistener -o AllReportableEntityNames -a
mqsireportproperties integrationNodeName -b httplistener -o HTTPListener -a
mqsireportproperties integrationNodeName -b httplistener -o HTTPSConnector  -a
```

Configuring an integration server to use SSL
If you want to use the integration server listener for HTTPS, configure values for the server's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: Specify a specific port on which the integration server listens for HTTPS requests, or leave the value unset to use the next available port number.

```
mqsichangeproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector
  -n explicitlySetPortNumber -v port_number
```

On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024. For the integration server to listen on these ports, the user ID under which the integration node is started must be root.

If you do not complete this step, the first available port in the default range (7843 - 7884) is used.

2. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector
  -n ReqClientAuth -v true
```

3. Optional: Change the SSL protocol.

The default protocol for the HTTPInput node is TLS. Run the following command to change it to SSL. The only supported versions of **TLSProtocols** are *TLSv1.2* and *TLSv1.3*. The values are not case-sensitive. If **TLSProtocols** is set to *all*, both versions are enabled. For more information, see Integration server HTTP listener parameters (SOAP and HTTP nodes). :

```
mqsichangeproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector
  -n TLSProtocols -v TLSv1.2
```

4. Restart the integration node after you change one or more of the listener properties.
5. Optional: Use the following command to display HTTPS properties:

```
mqsireportproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector  -r
```

Configuring SOAPRequest and SOAPAsyncRequest nodes to use SSL (HTTPS)

Configure the SOAPRequest and SOAPAsyncRequest nodes to communicate with other applications that use HTTP over SSL.

Configuring the nodes

Procedure

On the **HTTP Transport** page of the properties for the node, set the following properties:

1. In the `Web Service URL` property, type the URL of the web service you want to call.
2. Optional: Select the `Enable certificate revocation list checking` property.
3. Set other SSL properties as appropriate.

What to do next

Test your configuration.

Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)

Enable a message flow that uses HTTPInput and HTTPReply nodes to communicate with other applications that use HTTPS.

Before you begin

- Create the integration server to which you want to deploy the message flow. Follow the instructions in "Creating an integration server" on page 168.
- Decide which HTTP Listener you want to use for HTTPS messages. For more information about which listener to use for HTTPS messages, see "HTTP listeners" on page 795.
- Set up a public key infrastructure (PKI) to configure the keystores, truststores, passwords, and certificates to enable SSL communication. Follow the instructions in "Setting up a public key infrastructure" on page 2635. Following these instructions results in the integration server or integration node being configured for the PKI.

About this task

Complete these steps to enable a message flow that uses HTTPInput and HTTPReply nodes to communicate with other applications by using HTTP over SSL (HTTPS).

Procedure

1. Configure the integration server or integration node to use SSL.

   Complete one of the following substeps, depending on which HTTP listener you chose to use for HTTPS messages:

   - If you are using the integration node listener: Configure the integration node to use SSL
   - If you are using the integration server listener: Configure the integration server to use SSL

   Note: If you configure your integration node and integration servers such that the integration node listener is used for some integration servers, and the integration server listener for other integration servers, you must complete the first substep for the first set of integration servers and second substep for each integration server in the second set.
2. Create a message flow to use HTTPS
3. Test your configuration

Configuring an integration node to use SSL
If you want to use the integration node listener for HTTPS, configure values for the node's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: If you do not want to use the default port 7083 for HTTPS messages, specify the **port** on which the integration node listens:

   ```
   mqsichangeproperties integrationNodeName
     -b httplistener -o HTTPSConnector
     -n port -v Port_to_listen_on_for_https
   ```

   On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024.

   For the integration node to listen on these ports, the user ID under which the integration node is started must be root.
2. Optional: Enable Client Authentication (mutual authentication):

   ```
   mqsichangeproperties integrationNodeName -b httplistener -o HTTPSConnector
     -n ReqClientAuth -v true
   ```
3. Restart the integration node after you change one or more of the HTTP listener properties.
4. Optional: Use the following commands to display HTTP listener properties:

   ```
   mqsireportproperties integrationNodeName -b httplistener -o AllReportableEntityNames -a
   mqsireportproperties integrationNodeName -b httplistener -o HTTPListener -a
   mqsireportproperties integrationNodeName -b httplistener -o HTTPSConnector  -a
   ```

Configuring an integration server to use SSL
If you want to use the integration server listener for HTTPS, configure values for the server's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: Specify a specific port on which the integration server listens for HTTPS requests, or leave the value unset to use the next available port number.

   ```
   mqsichangeproperties integrationNodeName
     -e integration_server_name -o HTTPSConnector
     -n explicitlySetPortNumber -v port_number
   ```

   On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024. For the integration server to listen on these ports, the user ID under which the integration node is started must be root.

   If you do not complete this step, the first available port in the default range (7843 - 7884) is used.
2. Optional: Enable Client Authentication (mutual authentication):

   ```
   mqsichangeproperties integrationNodeName
     -e integration_server_name -o HTTPSConnector
     -n ReqClientAuth -v true
   ```

3. Optional: Change the SSL protocol.

   The default protocol for the HTTPInput node is TLS. Run the following command to change it to SSL. The only supported versions of **TLSProtocols** are *TLSv1.2* and *TLSv1.3*. The values are not case-sensitive. If **TLSProtocols** is set to *all*, both versions are enabled. For more information, see Integration server HTTP listener parameters (SOAP and HTTP nodes). :

   ```
   mqsichangeproperties integrationNodeName
     -e integration_server_name -o HTTPSConnector
     -n TLSProtocols -v TLSv1.2
   ```

4. Restart the integration node after you change one or more of the listener properties.
5. Optional: Use the following command to display HTTPS properties:

   ```
   mqsireportproperties integrationNodeName
     -e integration_server_name -o HTTPSConnector  -r
   ```

Creating a message flow to process HTTPS requests

About this task

You can create a simple message flow to use HTTPS by connecting an HTTPInput node to an HTTPReply node. The two most important properties to set on the HTTPInput node are:

- Path suffix for URL; for example, /* or /testHTTPS.
- Use HTTPS.

/* means that the HTTPInput node matches against any request that is sent to the HTTP listener on a designated port. This option is useful for testing purposes, but is not suitable for production systems.

You can now deploy the message flow to the broker. If you completed all the documented steps, message BIP3132 is written to the local system log (on Windows, the event log), stating that the HTTPS listener is started.

You can now test the system.

Testing your configuration

About this task

The simplest method of confirming the correct configuration of HTTPS, is to use a Web browser to make a request to the integration node over HTTPS.

Start a Web browser and enter the following URL:

```
https://localhost:7083/testHTTPS
```

Change values in the URL to reflect the changes that you made in your broker configuration; for example, the port number. When a window is displayed and you are asked to accept the certificate, select **Yes**. The browser refreshes the window and displays an empty HTML page:

- In Mozilla browsers, the empty HTML page looks like the following example:

  ```
  <html>
    <body/>
  </html>
  ```

- In Internet Explorer, the following information is displayed:

  ```
  XML document must have a top level element. Error processing resource
  'https://localhost:7083/testHTTPS'
  ```

These responses mean that a blank page was returned, indicating that the setup worked correctly. To add content to the empty page, you can add a Compute node to the flow.

You can use another HTTPS client to process HTTPS requests. Read the documentation for the client to find out how to configure it to make client connections over SSL.

You can also use another HTTPS client, such as a Java or .NET client, instead of the Web browser. Depending on the type of client, you might need to export the certificate (which was created with keytool) from the keystore file that is associated with the HTTP listener. Then, import it into the keystore for the client. Read the client documentation to find out how to configure the client to make client connections over SSL.

Configuring the HTTPRequest and HTTPAsyncRequest nodes to use SSL (HTTPS)

Configure the HTTPRequest or HTTPAsyncRequest nodes to communicate with other applications that use HTTP over SSL.

Before you begin

Set up a public key infrastructure (PKI) at integration node level: "Setting up a public key infrastructure" on page 2635

About this task

This topic describes the steps that you need to follow when configuring an HTTPRequest node on a Windows system. The steps that you must follow on other operating systems are almost identical.

To enable an HTTPRequest node to communicate using HTTP over SSL, an HTTPS server application is required. The information provided in this topic shows how to use the HTTPInput node for SSL as the server application, but the same details also apply when you are using any other server application. You specify the appropriate SSL client Authentication key alias field (Key Alias) on the HTTPRequest node or HTTPAsyncRequest node when the server key store contains multiple certificates for the server.

Complete the following sub-tasks:

1. "Creating a message flow to make HTTPS requests" on page 2622
2. "Testing your example" on page 2623.

Creating a message flow to make HTTPS requests

About this task

The following message flow creates a generic message flow for converting an IBM MQ message into an HTTP Request:

Procedure

1. Create a message flow with the nodes MQInput->HTTPRequest->Compute->MQOutput.
2. On the MQInput node, set the queue name to HTTPS.IN1 and create the IBM MQ queue.
3. On the MQOutput node, set the queue name to HTTPS.OUT1 and create the IBM MQ queue.
4. On the HTTPRequest node, set the web service URL to point to the HTTP server to call. To call the HTTPInput node, use `https://localhost:7083/testHTTPS`.
5. Optional: On the SSL settings properties tab, select `Enable certificate revocation list checking`.

   For more information on Certificate Revocation List (CRL) checking see, Working with certificate revocation lists.
6. On the Advanced properties tab of the HTTPRequest node, set the **`Response message location in tree`** property to *OutputRoot.BLOB*.
7. On the Compute node, add the following ESQL code:

```
CREATE COMPUTE MODULE test_https_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        -- CALL CopyMessageHeaders();
```

```
        CALL CopyEntireMessage();
        set OutputRoot.HTTPResponseHeader = null;
        RETURN TRUE;
    END;

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

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

What to do next

The message flow is now ready to be deployed to the integration node and tested.

Testing your example

About this task

To test that the example works, complete the following steps:

Procedure

1. Follow the instructions in "Configuring HTTPInput and HTTPReply nodes to use SSL (HTTPS)" on page 2619, including testing the example.
2. Deploy the HTTPRequest message flow.
3. Put a message to the IBM MQ queue HTTPS.IN1.

   If successful, a message appears on the output queue. If the process fails, an error appears in the local error log (which is the event log on Windows).

Securing the connection to CICS Transaction Server for z/OS by using SSL

Configure the CICSRequest node to communicate with CICS Transaction Server for z/OS over the Secure Sockets Layer (SSL) protocol by updating a CICS Connection policy or the CICSRequest node to use SSL.

Before you begin

Ensure that you have completed the following tasks:

1. The CICSRequest node does not support a separate truststore, so the keystore file must provide both personal and signer certificates. If client-authentication (CLIENTAUTH) is enabled in the TCPIPSERVICE in CICS, the IBM App Connect Enterprise keystore file must also contain a personal certificate that is trusted by CICS.
2. Define the COMMAREA data structure as a message set, as described in "Defining a CICS Transaction Server for z/OS data structure" on page 1186.
3. Configure IP InterCommunications (IPIC) protocol on CICS, as described in "Preparing the environment for the CICSRequest node" on page 1187.

About this task

To configure the CICSRequest node to use SSL, complete the following steps:

Procedure

1. For client-authenticated (CLIENTAUTH) SSL connections, CICS expects the SSL client certificate to be mapped to a RACF user ID. Therefore the SSL client certificate must be mapped to a RACF user ID before attempting to establish the SSL connection to CICS. If the client certificate is not mapped to a RACF user ID, IBM App Connect Enterprise might display a `ECI_ERR_NO_CICS` response. You can map a client certificate to a RACF user ID by using the RACF command **RACDCERT**, which stores the client certificate in the RACF database and associates a user ID with it, or by using RACF certificate name filtering. Client certificates can be mapped one-to-one with a user ID, or a mapping from one to the other can be provided to allow a many-to-one mapping. You can achieve this mapping by using one of the following methods:

   - **Associating a client certificate with a RACF user ID**

     a. Copy the certificate that you want to process into an MVS sequential file. The file must have variable length, blocked records (RECFM=VB), and be accessible from TSO.

     b. Run the **RACDCERT** command in TSO by using the following syntax:

     ```
     RACDCERT ADD('datasetname') TRUST [ ID(userid) ]
     ```

     Where:

     – *datasetname* is the name of the data set containing the client certificate.

     – *userid* is the user ID to be associated with the certificate. This parameter is optional. If omitted, the certificate is associated with the user issuing the **RACDCERT** command.

     When you issue the **RACDCERT** command, RACF creates a profile in the DIGTCERT class. This profile associates the certificate with the user ID. You can then use the profile to translate a certificate to a user ID without giving a password. For full details of RACF commands, see z/OS Security Server RACF Command Language Reference.

   - **RACF certificate name filtering**

     With certificate name filtering, client certificates are not stored in the RACF database. The association between one or more certificates and a RACF user ID is achieved by defining a filter rule that matches the distinguished name of the certificate owner or issuer (CA). A sample filter rule might look like the following example:

     ```
     RACDCERT ID(DEPT3USR) MAP SDNFILTER
     (OU=DEPT1.OU=DEPT2.O=IBM.L=LOC.SP=NY.C=US)
     ```

     This sample filter rule would associate user ID DEPT3USR with all certificates when the distinguished name of the certificate owner contains the organizational unit DEPT1 and DEPT2, the organization IBM, the locality LOC, the state/province NY, and the country US.

2. Turn on SSL support in the integration server by setting the `CICS Server` property on the CICS Connection policy.

   Alternatively you can configure the `CICS server` property directly on the CICSRequest node.

What to do next

When you have configured the integration server or the CICSRequest node to use SSL, develop a message flow that contains a CICSRequest node by following the steps in “Developing a message flow with a CICSRequest node” on page 1188.

SSL and the TCP/IP nodes

Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP nodes.

Configuring TCP/IP client nodes to use SSL
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP client nodes.

You can create or modify TCP/IP client connections that use SSL, by creating or modifying a policy. You can specify the type of protocol, and the allowed cipher suites. By default, SSL is not enabled for any policies.

Before you begin
Set up a public key infrastructure (PKI) at integration server level.

About this task

Follow these steps to configure the TCPIP nodes to use SSL:

1. "Changing a TCP/IP client configuration to use SSL" on page 2625
2. "Creating a TCP/IP client configuration that uses SSL" on page 2625

Changing a TCP/IP client configuration to use SSL

About this task

Use the Policy editor to change an existing TCPIP Client policy.

Procedure

1. Set the `SSL protocol` property to `TLS`.
2. Leave the `Cipher suites` property blank so that all available cipher suites can be used.
3. Deploy the policy project that contains your TCPIP Client policy to the integration server where you will deploy your associated message flow.

Creating a TCP/IP client configuration that uses SSL

About this task

Use the Policy editor to create a TCPIP Client policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).

Procedure

1. Set appropriate values for the `Port number` and `Host name` properties.
2. Set the `SSL protocol` property to `TLS`.
3. Set the Cipher suites property to a list of allowed cipher suites.
   For example, `SSL_RSA_WITH_RC4_128_MD5`.
4. Deploy the policy project that contains your TCPIP Client policy to the integration server where you will deploy your associated message flow.

Testing your configuration

About this task
Use either a TCPIPClientInput node, or a TCPIPClientOutput node to open a connection to a remote SSL server application that is listening on a TCP/IP port.

Configuring TCP/IP server nodes to use SSL
Configure a TCP/IP configuration to use SSL to secure connectivity to and from the TCPIP server nodes.

You can create or modify TCP/IP server connections that use SSL, by creating or modifying a policy. You can specify:

The type of protocol.
The allowed cipher suites.
A key alias.
Whether a connecting client should provide authentication information.

By default, SSL is not enabled for any policies.

About this task

Follow these steps to configure the TCPIP nodes to use SSL:

1. "Changing a TCP/IP server configuration to use SSL" on page 2626
2. "Creating a TCP/IP server configuration that uses SSL" on page 2626

Changing a TCP/IP server configuration to use SSL

About this task

Use the Policy editor to change an existing TCPIP Server policy.

Procedure

1. Set the `SSL protocol` property to TLS.
2. Leave the `Cipher suites` property blank so that all available cipher suites can be used.
3. Deploy the policy project that contains your TCPIP Server policy to the integration server where you will deploy your associated message flow.

Creating a TCP/IP server configuration that uses SSL

About this task

Use the Policy editor to create a TCPIP Server policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).

Procedure

1. Set the `Port number` property to an appropriate value.
2. Set the `SSL protocol` property to TLS.
3. Set the `Cipher suites` property to a list of allowed cipher suites, such as `SSL_RSA_WITH_RC4_128_MD5;SSL_RSA_WITH_3DES_EDE_CBC_SHA`.
4. Set the `SSL client authentication` property to `require` to indicate that connecting clients must authenticate.
5. Deploy the policy project that contains your TCPIP Server policy to the integration server where you will deploy your associated message flow.

Using an SSL key alias

About this task

A key alias identifies the key that is to be used for the SSL connection, if the keystore for your integration server contains more than one key. Specify the `SSL key alias` property on the TCPIP Server policy. The default value " " or `none`, means that an SSL key alias is not used. Any other string identifies the alias.

Note: If the keystore contains more than one key, and no key alias is defined, the Java virtual machine arbitrarily chooses a key at run time.

Procedure

1. Create a TCPIP Server policy (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Set the `Port number` property to the port number to use to make connections.
3. Set the `SSL protocol` property to TLS.
4. Specify a list of cipher suites to use (such as `SSL_RSA_WITH_RC4_128_MD5;SSL_RSA_WITH_3DES_EDE_CBC_SHA`).
5. Set the `SSL client authentication` property to require to indicate that connecting clients must authenticate.
6. Set the `SSL key alias` property to identify the key to be used.
7. Deploy the policy project that contains your TCPIP Server policy to the integration server where you will deploy your associated message flow.

Testing your configuration

About this task

To test your configuration, connect an SSL-enabled client, such as another program, or a web browser, to the server port. Connection error messages, such as handshake failures, or untrusted keys, indicate that you must change the configuration.

Client identity

About this task

If SSL client authentication is requested or required, and the client successfully authenticates, the distinguished name is present as an identity source token in the properties parser, in the tree propagated from the Open terminal at connection time. This applies only to the TCPIPServerInput node.

The `IdentitySourceToken` field is set to the distinguished name from the client certificate.
The `IdentitySourceType` field is set to the string `username`.
The `IdentitySourceIssuedBy` field is set to the issuer of the certificate presented by the client.

If SSL client authentication is requested, and the client does not provide the required credentials, the fields are set to blank.

Securing integration services by using SSL

You can secure integration services by configuring the SOAP/HTTP binding or JavaScript client API to use SSL and certificates.

Before you begin

- Create the integration server to which you want to deploy the integration service. Follow the instructions in "Creating an integration server" on page 168.
- Decide which HTTP Listener you want to use for HTTPS messages. For information about which listener to use for HTTPS messages, see "HTTP listeners" on page 795.
- Set up a public key infrastructure (PKI) to configure the keystores, truststores, passwords, and certificates to enable SSL communication. Follow the instructions in "Setting up a public key infrastructure" on page 2635. This results in the integration server or integration node being configured for the PKI.

About this task

Secure your integration services by configuring the integration node or integration server to use the PKI and SSL, and configuring the integration service bindings to use SSL.

Procedure

To secure your integration services by using SSL, complete the following steps:

1. Configure the integration server or integration node to use SSL.

   Complete one of the following substeps, depending on which HTTP listener you have chosen to use for HTTPS messages:

   - If you are using the integration node listener: Configure the integration node to use SSL
   - If you are using the integration server (embedded) listener: Configure the integration server to use SSL
2. Configure the integration service bindings to use SSL

Configuring an integration node to use SSL

If you want to use the integration node listener for HTTPS, configure values for the node's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: If you do not want to use the default port 7083 for HTTPS messages, specify the **port** on which the integration node listens:

   ```
   mqsichangeproperties integrationNodeName
     -b httplistener -o HTTPSConnector
     -n port -v Port_to_listen_on_for_https
   ```

   On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024.

   For the integration node to listen on these ports, the user ID under which the integration node is started must be root.
2. Optional: Enable Client Authentication (mutual authentication):

   ```
   mqsichangeproperties integrationNodeName -b httplistener -o HTTPSConnector
     -n ReqClientAuth -v true
   ```
3. Restart the integration node after you change one or more of the HTTP listener properties.
4. Optional: Use the following commands to display HTTP listener properties:

   ```
   mqsireportproperties integrationNodeName -b httplistener -o AllReportableEntityNames -a
   mqsireportproperties integrationNodeName -b httplistener -o HTTPListener -a
   mqsireportproperties integrationNodeName -b httplistener -o HTTPSConnector  -a
   ```

Configuring an integration server to use SSL

If you want to use the integration server listener for HTTPS, configure values for the server's HTTP listener properties.

About this task

Complete the following steps:

Procedure

1. Optional: Specify a specific port on which the integration server listens for HTTPS requests, or leave the value unset to use the next available port number.

   ```
   mqsichangeproperties integrationNodeName
   ```

```
  -e integration_server_name -o HTTPSConnector
  -n explicitlySetPortNumber -v port_number
```

On UNIX systems, only processes that run under a privileged user account (in most cases, root) can bind to ports lower than 1024. For the integration server to listen on these ports, the user ID under which the integration node is started must be root.

If you do not complete this step, the first available port in the default range (7843 - 7884) is used.

2. Optional: Enable Client Authentication (mutual authentication):

```
mqsichangeproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector
  -n ReqClientAuth -v true
```

3. Optional: Change the SSL protocol.

The default protocol for the HTTPInput node is TLS. Run the following command to change it to SSL. The only supported versions of **TLSProtocols** are *TLSv1.2* and *TLSv1.3*. The values are not case-sensitive. If **TLSProtocols** is set to *all*, both versions are enabled. For more information, see Integration server HTTP listener parameters (SOAP and HTTP nodes). :

```
mqsichangeproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector
  -n TLSProtocols -v TLSv1.2
```

4. Restart the integration node after you change one or more of the listener properties.
5. Optional: Use the following command to display HTTPS properties:

```
mqsireportproperties integrationNodeName
  -e integration_server_name -o HTTPSConnector  -r
```

Configuring the integration service bindings to use SSL

About this task

Configure the integration service bindings to use SSL by completing the following steps:

Procedure

1. In the IBM App Connect Enterprise Toolkit, open your integration service in the integration service editor by double-clicking **Integration Service Description** in the Application Development view.
2. Click the **Service** tab.

   The integration service description is displayed, which includes the integration service bindings.
3. If you are using the SOAP/HTTP binding, then click **SOAP/HTTP Binding** and select **Use HTTPS** from the HTTP Transport properties panel.
4. If you are using the JavaScript client API, then click **JavaScript client API** and then select **Use HTTPS** from the Basic properties panel.

   Note: If you are using a web browser-based JavaScript application to call the integration service, then you must select **Use HTTPS** on both the SOAP/HTTP binding and the JavaScript client API. The HTTP proxy servlet routes requests only to endpoints that use the same protocol as the web browser. The HTTP proxy servlet routes requests to both the SOAP and JavaScript client API endpoints, and so both endpoints must match the web browser protocol.
5. Save and redeploy the integration service.

Results

You have configured the integration service to use SSL.

Checking the password for a resource that is used by an integration server

To check what security credentials are set on an integration server that is connected to a remote system or database, use the **mqsireportdbparms** command.

About this task

If you have an integration server that accesses external resources, you can check what security credentials are set if a security identity was created for the integration server. For more information, see “Security identities for the integration server connecting to external systems” on page 2582.

You can check whether a password is valid for a set of security credentials. You might want to check a password if you recently updated the password while the integration server was running, to see whether the new password is in effect.

To review the security credentials that are set for a resource that is connected to an integration serverintegration server, complete the following steps.

Procedure

Run the **mqsireportdbparms** command with the work directory of the integration server that you want to check in the following format:

- If you are connecting to an SFTP data source:
- If you are connecting to a non-SFTP data source:

```
mqsireportdbparms -w workDir  userdb -u user_id -p password
```

Where the names represent:

- *workDir* is the work directory of the integration server
- *server* as the details of your SFTP connection
- *SSH_identity* as the SSH connection name
- *userdb* as the data source name
- *user_id* as the user ID that you want to check
- *password* as the password that you want to check is valid for the integration server

Results

A result is returned that confirms whether the password that was provided is correct, or if it was incorrect. If there are no security credentials set for that integration node, then an information message is displayed.

Additionally, the command returns if the integration server was restarted after a password change, so that you know what password is valid for the integration server while it is running.

Diagnosing security problems

This topic explains how to find out why access to a secured flow has been denied.

About this task

By default, security exceptions occurring in an input node are not processed in the same way as other errors (see “Security exception processing” on page 2582). Security exceptions are not logged to the system event log, to prevent a security denial of service attack filling the logs and destabilizing the system.

This means that, by default, you cannot diagnose input node security exceptions in the same way as other errors. However, in a SecurityPEP node, a failing security operation causes a security exception to be

raised, wrapped in a normal recoverable exception, which invokes the error handling that is provided by the message flow.

To see what might be causing the security exceptions, you can do either of the following things:

- Select the **Treat Security Exceptions as normal exceptions** property on the input nodes.
- Use the user trace.

The following steps show you how to use the user trace to find out why access to a secured message flow has been denied:

Procedure

1. Use the **mqsireloadsecurity** command to clear the security cache, so that the traced request goes to the security provider rather than using a result held in the cache. This ensures that the reason codes returned from the security provider are displayed in the traced exception.
2. Enable user trace (see "User trace example" on page 3030 for more information).
3. Resend the request that has been rejected by the security provider.
4. Stop the user trace.
5. Examine the trace information that was recorded by the user trace. This trace information contains the error codes provided by the integration server and the security provider.

Permitting web browsers to access deployed HTTP services by enabling Cross-Origin Resource Sharing

You can configure IBM App Connect Enterprise to permit cross-origin requests from a web browser by enabling Cross-Origin Resource Sharing (CORS).

About this task

IBM App Connect Enterprise includes full support for CORS according to the specification that is available online at Cross-Origin Resource Sharing. You can configure the HTTP listeners for an integration server and the HTTP listeners for an integration node so that they respond to and permit cross-origin requests from a web browser.

Procedure

If you want to permit a web page that is running in a web browser to make HTTP requests to HTTP services that are deployed to IBM App Connect Enterprise, where those HTTP services are being hosted on an HTTP listener for an integration server or an HTTP listener for the integration node, complete the following steps:

1. Enable CORS on the HTTP listeners. The connectorName for the HTTP listener is HTTPConnector, and for the HTTPS listener is HTTPSConnector. If both HTTP and HTTPS are in use, you must configure the HTTP listeners for HTTP and HTTPS independently.

   The following example shows how to configure the integration server HTTP listener:

   ```
   mqsichangeproperties integrationNodeName -e integrationServerName -o connectorName -n
    corsEnabled -v true
   ```

   The following example shows how to configure the integration node HTTP listener:

   ```
   mqsichangeproperties integrationNodeName -b httplistener -o connectorName -n corsEnabled -v
    true
   ```

2. Optional: The default values for the CORS properties for the HTTP listener are permissive. All origins and HTTP methods are permitted by default. You can restrict the permitted origins and HTTP methods for cross-origin requests by configuring other CORS properties.

3. Optional: To display the currently configured CORS properties on the HTTP listener, run the following command:

   For the integration server HTTP listener:

   ```
   mqsireportproperties integrationNodeName -e integrationServerName -o connectorName -r
   ```

   For the integration node HTTP listener:

   ```
   mqsireportproperties integrationNodeName -b httplistener -o connectorName -r
   ```

Results

Cross-Origin Resource Sharing is now enabled. The web page that is running in a web browser is now permitted to make HTTP requests to HTTP services that are deployed to IBM App Connect Enterprise, where those HTTP services are being hosted on an HTTP listener for an integration server or an HTTP listener for the integration node.

What to do next

When CORS properties are modified on an HTTP listener for the integration server or an HTTP listener for the integration node, the changes take effect immediately. You are not required to restart any of the deployed HTTP services, the integration server, or the integration node.

Complete the following checks:

- Test the configuration by making HTTP requests from a web page that is running in a web browser to the deployed HTTP services that are being hosted on the HTTP listener that you configured.
- Check that the HTTPInput node in the message flow is called when the request is made, and that the message flow processes the request as expected.
- Check that the response that is sent through the HTTPReply node in the message flow is returned to the web page that made the request, and that the response is not dropped by the web browser.
- Check that this processing works for all the HTTP methods that the web page makes requests for.

Security for runtime components

You must consider several security aspects when you are configuring integration nodes or integration servers running on Windows or Linux platforms.

Before you begin

- Read “Security overview” on page 2491 for an introduction to various aspects of security.
- Refer to “Planning for security” on page 17, which contains links to security information that you need before, during, and after installation of IBM App Connect Enterprise.

About this task

Use the following list of tasks as a security checklist:

- “Creating user IDs” on page 2633
- “Controlling access to IBM App Connect Enterprise” on page 2633
- “Implementing SSL authentication” on page 2635

Creating user IDs

When you plan the administration of your integration node configuration, you might have to define one or more user IDs for the tasks associated with particular roles.

About this task

Some operating systems, and other products, impose restrictions on user IDs:

- On Windows systems, user IDs can be up to 12 characters long, but on Linux, UNIX, and z/OS systems, they are restricted to eight characters.
- Database products might also restrict user IDs to eight characters; for example DB2 has this restriction. If you have a mixed environment, ensure that the user IDs that you use are limited to a maximum of eight characters.
- Ensure that the case (upper, lower, or mixed) of user IDs is consistent. In some environments, uppercase and lowercase user IDs are considered the same, but in other environments, user IDs of different case are considered unique.

  For example, on Windows systems, the user IDs 'tester' and 'TESTER' are identical; on Linux and UNIX systems, they are recognized as different user IDs.
- Check the validity of spaces and special characters in user IDs to ensure that, if used, these characters are accepted by all relevant systems and products that you install.

Consider the following roles:

Procedure

- Administrator user IDs that can issue **mqsi*** commands.

  Ensure that all these user IDs are members of the mqbrkrs group.

  If you enable integration node administration security, you must set up additional authorization for user IDs for the commands listed in Commands and authorizations for administration security. For more information about integration node administration security, see “Administration security overview” on page 2497.
- On Windows only, service user IDs under which integration nodes run.

  For further information, see “Deciding which user account to use for the integration node service ID ” on page 2634.
- IBM App Connect Enterprise Toolkit users.

  See “Security for the IBM App Connect Enterprise Toolkit ” on page 2648 for information about checking and securing connections from the IBM App Connect Enterprise Toolkit.

Controlling access to IBM App Connect Enterprise

There are several factors to consider when you are deciding which users can execute App Connect Enterprise commands, and which users can control security for your resources.

About this task

Although most security for App Connect Enterprise and its resources is optional, you might find it appropriate to restrict the tasks that some user IDs can perform. You can then apply greater control to monitor changes.

You can control all IBM App Connect Enterprise administration tasks by enabling administration security. You can enable administration security and specify the authorization mode, either by setting the **authorizationEnabled** and **authorizationMode** properties in the `node.conf.yaml` or `server.conf.yaml` configuration files, or by using the **mqsichangeauthmode** command. This task is described in “Enabling administration security” on page 2504, and is independent of the tasks described in this section.

When you are deciding which users are to perform the different tasks, consider the following steps:

Procedure

1. “Deciding which user account to use for the integration node service ID ” on page 2634
2. “Setting security on the integration node” on page 2635
3. “Securing the integration node registry” on page 2635

Deciding which user account to use for the integration node service ID

About this task

On a Linux or UNIX operating system, when you run the **mqsistart** command with a user ID that is a member of the mqm and mqbrkrs groups, the user ID under which you run the **mqsistart** command becomes the user ID under which the integration node process runs.

On the Windows platform, the integration node runs under a service user account. To decide which user ID to use for the service ID, answer the following questions:

Procedure

1. Do you want the integration node to run under a Windows local account?
   a) No: Go to the next question.
   b) Yes: Ensure that your user ID has the following characteristics:
      - It is defined in your local domain.
      - It is a member of the mqbrkrs group.

      Go to “Setting security on the integration node” on page 2635.
2. Do you want the integration node to run under a Windows domain account?
   a) No: Go to the next question.
   b) Yes: Assume that your computer named, for example, WKSTN1, is a member of a domain named DOMAIN1. When you run an integration node using, for example, DOMAIN1\user1, ensure that:
      - Your user ID has been granted the `Logon as a service` privilege (from the Local Security Policy).
      - DOMAIN1\user1 is a member of DOMAIN1\MyDomainGroup group, where MyDomainGroup is a domain group which you have defined on your domain controller.
      - DOMAIN1\MyDomainGroup is a member of WKSTN1\mqbrkrs.

      Go to “Setting security on the integration node” on page 2635.
3. Do you want the integration node to run under theWindows built-in `LocalSystem` account?
   a) Yes: Specify `LocalSystem` for the *-i* parameter on the **mqsicreatebroker** command.

      In either case you must enter the *-a* (password) parameter on the command line, but the value entered is ignored.

Results

Note that for cases one and two above, the user ID chosen must be granted the `Logon as a service` privilege.

This is normally done automatically by the **mqsichangeproperties** command when a service user ID is specified that does not have this privilege.

However, if you want to do this manually before running these commands, you can do this by using the Local Security Policy tool in Windows, which you can access by selecting **Control Panel** > **Performance and maintenance** > **Administrative Tools** > **Local Security Policy**.

Setting security on the integration node

About this task

If you are using the queue-based authorization mode for the integration node (*mq* mode), the local mqbrkrs group is granted access to internal queues whose names begin with the characters SYSTEM.BROKER. If you are using the file-based authorization mode, the local mqbrkrs group is granted read, write, and execute permissions on the integration node for running local mqsi commands. Ensure that user IDs requiring these permissions are members of the mqbrkrs group.

Securing the integration node registry

About this task

Integration node operation depends on the information in the integration node registry, which you must secure to guard against accidental corruption. The integration node registry is stored on the file system under the work path directory, which is specified by the MQSI_WORKPATH environment variable. Set your operating system security options so that only user IDs that are members of the group mqbrkrs can read from or write to *integrationNodeName*/`CurrentVersion` and all subkeys.

Implementing SSL authentication

Use SSL authentication to enhance security in your integration node environment.

About this task

The following topics contain instructions for implementing SSL authentication:

- "Setting up a public key infrastructure" on page 2635
- "Authorization by using SSL Client Certificates" on page 2643
- "Securing a REST API by using HTTPS" on page 2041
- "Securing message flows by using SSL" on page 2615
- "Securing integration services by using SSL" on page 2627

Setting up a public key infrastructure

Configure keystores, truststores, passwords, and certificates to enable SSL communication, and web services security.

Before you begin

- For an overview of encryption and public key infrastructure (PKI), see "Public key cryptography" on page 2492.
- Make sure you understand the purpose of certificates, keystores, and truststores; see "Digital certificates" on page 2493.
- Decide how you want to use the PKI. You can configure keystores and truststores at the following levels:
  - Integration node level (one keystore, and one truststore for each integration node).
  - Integration node listener level (one keystore and one truststore for each listener in an integration node). Integration node listeners that do not have a PKI configured use the integration node level PKI configuration.
  - Integration server level (one keystore and one truststore for each integration server). For integration servers under an integration node, if a PKI is not configured at the integration server level, the integration node level PKI configuration is used.
  - Integration server listener level (one keystore and one truststore for each listener in an integration server). Integration server listeners that do not have a PKI configured use the integration server level

PKI configuration. For integration servers under an integration node, if a PKI is not configured at either the integration server listener level or the integration server level, the integration node level PKI configuration is used.

Encryption strength

The IBM App Connect Enterprise Java runtime environment (JRE) is provided with strong but limited strength encryption. If you cannot import keys into keystores, limited strength encryption might be the cause. Either start ikeyman by using the **strmqikm** command, or download unrestricted jurisdiction policy files from IBM developer kits: Security information.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before you download or use the unrestricted policy files, you must check the laws of your country. Check its regulations and its policies on the import, possession, use, and re-export of encryption software, to determine whether it is permitted. Note that when you apply a fix pack to an existing IBM App Connect Enterprise installation, the JVM is overwritten, including any updated policy set files. These policy set files must be restored before you restart the integration node.

IBM App Connect Enterprise currently supports up to 4096 bit keys. Larger keys require more CPU resources for encryption and decryption.

About this task

This topic uses the command line tool **gsk8capicmd_64** to create and populate keystores and truststores. The **gsk8capicmd_64** tool is a part of the Global Secure Toolkit, supplied with IBM MQ and located in the `gskit8\bin` subdirectory of the IBM MQ installation directory. For example, on Windows the **gsk8capicmd_64** tool is in the `C:\Program Files\IBM\MQ\gskit8\bin` directory.

The IBM App Connect Enterprise JVM also supplies other options, for example:

- A command-line tool, **keytool**.
- A graphical tool, **iKeyman**.

To create the infrastructure, complete the following tasks:

1. "Creating a keystore file or a truststore" on page 2636
2. Either "Creating a self-signed certificate for test use" on page 2637 or "Importing a certificate for production use" on page 2637
3. "Viewing details of a certificate" on page 2638
4. "Extracting a certificate" on page 2638
5. "Adding a signer certificate to the truststore" on page 2639
6. "Listing all certificates in a keystore" on page 2640
7. "Configuring PKI at integration node level" on page 2640
8. "Configuring PKI for the integration node HTTP listener" on page 2641
9. "Configuring PKI at an integration server level" on page 2642

Creating a keystore file or a truststore

The keystore file contains the personal certificate for the integration node or for the integration server. You can have only one personal certificate in the keystore. You can store signer certificates in the same file, or create a separate file, which is known as a truststore.

Procedure

1. Set the PATH environment variable to point to the GSKit library directory.
   For example,

```
set PATH=C:\Program Files\IBM\MQ\gskit8\bin;C:\Program Files\IBM\MQ\gskit8\lib;%PATH%
```

2. Issue the following command:

```
gsk8capicmd_64 -keydb -create
  -db keystore_name
  [-pw password]
  -type jks
```

The *password* argument is optional. If you omit it, you are prompted to enter a password.

For example:

```
gsk8capicmd_64 -keydb -create
  -db myBrokerKeystore.jks
  -type jks
A password is required to access this key database.
Please enter a password:
```

Creating a self-signed certificate for test use

Use self-signed certificates only for testing SSL, not in production.

Procedure

Enter the following command:

```
gsk8capicmd_64 -cert -create
  -db keystore_name
  [-pw password]
  -label cert_label
  -dn "distinguished_name"
```

For example:

```
gsk8capicmd_64 -cert -create
  -db myBrokerKeystore.jks
  -label MyCert
  -dn "CN=MyBroker.Server,O=IBM,OU=ISSW,L=Hursley,C=GB"
A password is required to access this key database.
Please enter a password:
```

Importing a certificate for production use

Import a personal certificate from a certificate authority for production use.

Procedure

Issue the following command:

```
gsk8capicmd_64 -cert -import
  -db pkcs12_file_name
  [-pw pkcs12_password]
  -label label
  -type pkcs12
  -target keystore_name
  [-target_pw keystore_password]
```

If you are going to use the keystore for inbound https connections, then ensure that you always specify a **target_pw**, and that it matches the password required to access the key database (keystore).

For example:

```
gsk8capicmd_64 -cert -import
  -db SOAPListenerCertificate.p12
  -label soaplistener
  -type pkcs12
  -target myBrokerKeystore.jks
  -target_pw  myBrokerKpass
A password is required to access this key database.
Please enter a password:
```

Viewing details of a certificate

View details of a certificate to verify that they are correct and appropriate for your needs.

Procedure

Issue the following command:

```
gsk8capicmd_64 -cert -details
  -db keystore_name
  [-pw password]
  -label label
```

For example:

```
gsk8capicmd_64 -cert -details
  -db myKeyStore.jks
  -label MyCert
A password is required to access this key database.
Please enter a password:

Label: MyCert
Key Size: 1024
Version: X509 V3
Serial Number: 4A D7 39 1F
Issued By: MyBroker.Server
ISSW
IBM
Hursley, GB
Subject: MyBroker.Server
ISSW
IBM
Hursley, GB
Valid From: 15 October 2018 16:00:47 o'clock BST To: 15 October 2020 16:00:47 o'
clock BST
Fingerprint: 98:5D:C4:70:A0:28:84:72:FB:F6:3A:D2:D2:F5:EE:8D:30:33:87:82
Signature Algorithm: 1.2.840.113549.1.1.4
Trust Status: enabled
```

Extracting a certificate

Generate a copy of a self-signed certificate that you can import as a trusted (or signer certificate) into a truststore file. Use this procedure only for testing, not production.

About this task

Certificates can be extracted in two formats:

- Base64-encoded ASCII data (`.arm`). This format is convenient for inclusion in XML messages, and transmission over the Internet.
- Binary DER data (`.der`).

Procedure

Issue the following command:

```
gsk8capicmd_64 -cert -extract
  -db keystore_name
  -pw keystore_passwd
  -label label
  -target file_name
  [-format ascii | binary]
```

For example:

```
gsk8capicmd_64 -cert -extract
  -db myBrokerKeystore.jks
  -pw myKeyPass
  -label MyCert
  -target MyCert.arm
  -format ascii
```

You can then view the certificate in a text editor, such as Notepad:

```
notepad MyCert.arm
-----BEGIN CERTIFICATE-----
MIICIzCCAYygAwIBAgIEStc5HzANBgkqhkiG9w0BAQQFADBWMQswCQYDVQQGEwJHQjEQMA4GA1UE
BxMHSHVyc2xleTEMMAoGA1UEChMDSUJNMQ0wCwYDVQQLEwRJU1NXMRgwFgYDVQQDEw9NeUJyb2tl
ci5TZXJ2ZXIwHhcNMDkxMDE1MTUwMDQ3WhcNMTAxMDE1MTUwMDQ3WjBWMQswCQYDVQQGEwJHQjEQ
MA4GA1UEBxMHSHVyc2xleTEMMAoGA1UEChMDSUJNMQ0wCwYDVQQLEwRJU1NXMRgwFgYDVQQDEw9N
eUJyb2tlci5TZXJ2ZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMwkK5kFLwC29YsHLXlf
hd0CgqFeytHlI0sZesdi8hEPXKsOzs3OQta2b0GZyUbBkh4tNeUHNWE9o7Hx2/SfziPQRKUw908R
F/6FPaHGezRkkaLJGX3uEhjt/2+n5tOJGytnKWaWJTpzdmZ79c0XjFvO83q3yXPYjKzq8rS1iVBf
AgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAQEjpvZkjRcg3AHqY4RWbSMtXVWFFyoHSbjymR8IdURoQ
DCGZ2jsv3kxQLADaCXOBYgohGJAHS7PzkQoHUCiHR0kusyuAt1MNYbhEcs+BYAzvsSz1ay4oiqCw
Qs3aeNLVOb9c1RyzbuKYZl0uX59GAfGVLvyk6vQ/g7wPVL4TVgc=
-----END CERTIFICATE-----
```

Adding a signer certificate to the truststore

Add a signer certificate to the truststore of an integration node or integration server.

About this task

The following steps show how to add an extracted certificate as signer certificate to the truststore file. Adding the integration node self-signed certificate to an integration node or integration server truststore enables request nodes (HTTP or SOAP) to send test messages to input nodes (HTTP or SOAP) when the flows are running on the integration node or integration server.

Procedure

Issue the following command:

```
gsk8capicmd_64 -cert -add
  -db truststore_name
  [-pw password]
  -label label
  -file file_name
  -format [ascii | binary]
```

For example:

```
gsk8capicmd_64 -cert -add
  -db myBrokerTruststore.jks
  -label CACert
  -file TRUSTEDPublicCerticate.arm
  -format ascii
```

You can view details of the certificate:

```
gsk8capicmd_64 -cert -details -db myBrokerTruststore.jks -label CACert
A password is required to access this key database.
Please enter a password:

Label: CACert
Key Size: 1024
Version: X509 V3
Serial Number: 49 49 23 1B
Issued By: VSR1BK
ISSW
IBM
GB
Subject: VSR1BK
ISSW
IBM
GB
Valid From: 17 December 2008 16:04:43 o'clock GMT To: 17 December 2009 16:04:43
o'clock GMT
Fingerprint: CB:39:E7:D8:1D:C0:00:A1:3D:B1:97:69:7A:A7:77:19:6D:09:C2:A7
Signature Algorithm: 1.2.840.113549.1.1.4
Trust Status: enabled
```

Listing all certificates in a keystore

Procedure

Issue the following command:

```
gsk8capicmd_64 -cert -list
  -db keystore_name
```

For example:

```
gsk8capicmd_64 -cert -list
  -db myBrokerKeystore.jks
A password is required to access this key database.
Please enter a password:

Certificates in database: myBrokerKeystore.jks
    verisign class 1 public primary certification authority - g3
    verisign class 4 public primary certification authority - g3
    verisign class 1 public primary certification authority - g2
    verisign class 4 public primary certification authority - g2
    verisign class 2 public primary certification authority
    entrust.net global client certification authority
    rsa secure server certification authority
    verisign class 2 public primary certification authority - g3
    verisign class 2 public primary certification authority - g2
    verisign class 3 secure server ca
    verisign class 3 public primary certification authority
    verisign class 3 public primary certification authority - g3
    verisign class 3 public primary certification authority - g2
    thawte premium server ca
    verisign class 1 public primary certification authority
    entrust.net global secure server certification authority
    thawte personal basic ca
    thawte personal premium ca
    thawte personal freemail ca
    verisign international server ca - class 3
    thawte server ca
    entrust.net certification authority (2048)
    cacert
    entrust.net client certification authority
    entrust.net secure server certification authority
    soaplistener
    mycert
```

Configuring PKI at integration node level

Define the integration node registry properties that identify the location, name, and password of the keystore and truststore files.

About this task

These settings are used as the default settings for the integration node HTTP listener and all embedded HTTP listeners in integration servers on the integration node. These settings can be overridden for the integration node HTTP listener (see “Configuring PKI for the integration node HTTP listener” on page 2641) and for individual embedded HTTP listeners (see “Configuring PKI at an integration server level” on page 2642).

1. Start the integration node:

   ```
   mqsistart integrationNodeName
   ```

2. Display the current settings of the integration node registry properties:

   ```
   mqsireportproperties integrationNodeName
     -o BrokerRegistry
     -r
   ```

3. Set the keystore property:

   ```
   mqsichangeproperties integrationNodeName
     -o BrokerRegistry
   ```

```
  -n brokerKeystoreFile
  -v install_dir\MyBrokerKeystore.jks
```

4. Set the truststore property:

```
mqsichangeproperties integrationNodeName
  -o BrokerRegistry
  -n brokerTruststoreFile
  -v install_dir\MyBrokerTruststore.jks
```

5. Stop the integration node:

```
mqsistop integrationNodeName
```

6. Set the password for the keystore:

```
mqsisetdbparms integrationNodeName
  -n brokerKeystore::password
  -u ignore
  -p keystore_pass
```

7. Set the password for the truststore:

```
mqsisetdbparms integrationNodeName
  -n brokerTruststore::password
  -u ignore
  -p truststore_pass
```

8. Start the integration node:

```
mqsistart integrationNodeName
```

9. Display and verify the integration node registry properties:

```
mqsireportproperties integrationNodeName
  -o BrokerRegistry
  -r
```

Configuring PKI for the integration node HTTP listener

Define the properties for the integration node HTTP listener to identify the location, name, and password of the keystore and truststore files.

About this task

These settings override any PKI configuration that is set at the integration node level. If you enable SSL on the integration node HTTP listener but do not set the following properties, then the integration node-level settings are applied, see “Configuring PKI at integration node level” on page 2640.

1. Start the integration node.

```
mqsistart integrationNodeName
```

2. Display the current settings of the integration node listener properties.

```
mqsireportproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -a
```

3. Set the keystore property.

```
mqsichangeproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -n keystoreFile
  -v install_dir\MyBrokerKeystore.jks
```

4. Set the truststore property.

```
mqsichangeproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -n truststoreFile
  -v install_dir\MyBrokerTruststore.jks
```

5. Set the password for the keystore.

```
mqsichangeproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -n keystorePass
  -v keystore_pass
```

6. Set the password for the truststore.

```
mqsichangeproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -n TruststorePassword
  -v truststore_pass
```

7. Display and verify the integration node listener properties.

```
mqsireportproperties integrationNodeName
  -b NodeHttpListener
  -o HTTPSConnector
  -a
```

Configuring PKI at an integration server level

Define the properties for the integration server HTTP listener to identify the location, name, and password of the keystore and truststore files.

About this task

These settings override any PKI configuration that is set at the integration node level.

If you enable SSL for the integration server HTTP listener but do not set the following properties, then the integration node-level settings are applied, see “Configuring PKI at integration node level” on page 2640.

1. Start the integration node.

```
mqsistart integrationNodeName
```

2. Display the current settings of the ComIbmJVMManager properties.

```
mqsireportproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -r
```

3. Set the keystore property.

```
mqsichangeproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -n keystoreFile
  -v install_dir\MyExecGrpKeystore.jks
```

4. Set the keystore password key property. The value for this property is in the format *any_prefix_name*`::password`. This value is used to correlate the password that is defined in the **mqsisetdbparms** command.

```
mqsichangeproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -n keystorePass
  -v brokerKeystore::password
```

5. Set the truststore property.

```
mqsichangeproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -n truststoreFile
  -v install_dir\MyExecGrpTruststore.jks
```

6. Set the truststore password key property. The value for this property is in the format *any_prefix_name*::password. This value is used to correlate the password that is defined in the **mqsisetdbparms** command.

```
mqsichangeproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -n truststorePass
  -v brokerTruststore::password
```

7. Stop the integration node.

```
mqsistop integrationNodeName
```

8. Set the password for the keystore.

```
mqsisetdbparms integrationNodeName
  -n brokerKeystore::password
  -u ignore
  -p keystore_pass
```

9. Set the password for the truststore.

```
mqsisetdbparms integrationNodeName
  -n brokerTruststore::password
  -u ignore
  -p truststore_pass
```

10. Start the integration node.

```
mqsistart integrationNodeName
```

11. Display and verify the ComIbmJVMManager properties.

```
mqsireportproperties integrationNodeName
  -e exec_grp_name
  -o ComIbmJVMManager
  -r
```

For information about cipher-suite requirements (such as the cryptographic algorithm and corresponding key lengths), see the Java Secure Socket Extension (JSSE) IBMJSSE2 Provider reference guide.

Authorization by using SSL Client Certificates

Client authentication data for SSL X509 certificates can be propagated into the local environment and used for authorization.

When you implement a message flow to use SSL authentication, you can check authenticated client certificates for authorization. When a security profile is configured for authorization and set on the input node, the data is passed to a security manager. An integration node security manager receives relevant parts of the certificate for authorization and sends it to the properties parser. During authentication, data from the identity or security token that is provided replaces the values in the properties tree identity fields. This data can be from a Basic-Auth transport header or a WS-Security token, for example. Parameter data in the certificate replaces fields in the properties tree. The following fields are reset with new data:

- The **IdentitySourceType** is set to the user name.
- The **IdentitySourceToken** is set to the subject name of the client certificate.
- The **IdentitySourceIssuedBy** is set to the issuer name of the client certificate.

When you use the SOAPInput, HTTPInput, SCAInput, or TCPIPServerInput nodes, properties tree fields contain the information from the client certificate. Propagation is not automatically enabled but when it is enabled, a certificate is processed throughout the message flow and propagated for output or request nodes. By populating the local environment, the certificate data becomes available to the rest of the message flow.

A higher level of authentication (such as Basic-Auth or WS-Security) can overwrite the properties tree. Because of missing properties tree data, you are unable to authorize the client at the input node. However, you can use a SecurityPEP node to locate authentication (or other certificate) fields in the local environment to do the authorization. You can locate client certificates by using the local variable `LocalEnvironment.`*`input_node_name`*`.Input.TransportSecurity.ClientAuth.Certificate`, where *input_node_name* is one of SOAP, HTTP, or TCPIP.

Different nodes access the client certificate in different ways:

- Use a SecurityPEP node for authorization of the **`subject`** field in a client certificate that is received by a SOAPInput node. Specify the following XPath in the `identity token location` property on a SecurityPEP node:

  ```
  $LocalEnvironment/SOAP/Input/TransportSecurity/ClientAuth/Certificate/Subject
  ```

  For more information about the SecurityPEP node, see node.
- Use a JavaCompute node to retrieve the **`issuer`** field in a client certificate that is received by an HTTPInput node.

  ```
  String clientCertSubject = localEnv.getFirstElementByPath("HTTP/Input/TransportSecurity/
  ClientAuth/Certificate/Issuer").getValueAsString();
  ```

  For more information about the JavaCompute node, see node.
- Use a Compute node to set the **`subject`** field of a certificate that is received by a TCPIPServerInput node.

  ```
  SET LocalEnvironment.TCPIP.Input.TransportSecurity.ClientAuth.Certificate.Subject =
   'BROKERUSER';
  ```

  For more information about the Compute node, see node.

SSL cipher suites

A cipher suite is a collection of algorithms that are used to encrypt data.

During SSL authentication, the client and server compare cipher suites and select the first one that they have in common. If no suitable cipher suites exist, the server returns a handshake failure alert and closes the connection.

To use SSL encryption, you need a Java Secure Socket Extension (JSSE) provider. IBM App Connect Enterprise supports the ciphers that are provided by the JSSE provider. For more information about Java security, see IBM Java security web page.

For a list of all the cipher suites that are supported by IBM App Connect Enterprise, see Appendix A of the IBM JSSE2 Guide.

SSL and Kerberos support for SQL server for Linux and UNIX clients

Connecting Microsoft SQL Server with SSL protection and Kerberos authentication to propagate user ID without the need to reenter the password multiple times.

Secure Sockets Layers (SSL) is a standard that provides confidentiality and integrity to a TCP/IP transport, such as the one that the ODBC driver uses to communicate with an SQL Server instance.

Kerberos is a protocol that is devised by MIT to allow resources to be secured through a trusted third party (The Kerberos KDC). When the protocol is implemented by Microsoft, it can be used only to secure access within a complete active directory domain.

The tasks in the following topics configure the product so that at runtime IBM App Connect Enterprise completes the following actions:

1. Connects to the directory server
2. Retrieves the TGT (Ticket-Granting Ticket)
3. Store the TGT in the credential cache
4. Update the environment to point to the credential cache
5. Pass control to the DataDirect driver

These tasks require the following things:

- A Linux or UNIX client machine
- An Active Directory server
- An SQL server

Configuring IBM App Connect Enterprise to connect to SQL server: Part 1

The following task demonstrates the initial setup of the `odbc.ini` and `krb5.conf` files.

About this task

These tasks are required to connect your IBM App Connect Enterprise machine to SQL server without using SSL. There are some initial tasks to configure the `odbc.ini` and `krb5.conf` files before you begin:

Procedure

1. Configure the Linux or UNIX client machine to connect to the SQL server by adding the following entry to your `odbc.ini` file:

```
[TESTDB]
Driver=/opt/IBM/mqsi/11.0.0.n/server/ODBC/drivers/lib/UKsqls95.so
Description=DataDirect SQL Server Wire Protocol
Database=TESTDB
HostName=sqlserver.domain.company.com
PortNumber=1433
AnsiNPW=1
LoginTimeout=0
QueryTimeout=0
;# To specify a named instance of SQLServer replace the HostName and PortNumber lines with
;#HostName=sqlserver.domain.company.com\named_instance
;# To use Integrated Windows Authentication uncomment the following lines.
;#AuthenticationMethod=9
;#Domain=domain.company.com
```

   In this example:

   - *TESTDB* is the name of a database.
   - *sqlserver.domain.company.com* is the address of your SQL server.
   - *1433* is the port number of your SQL server.

     To use a named instance instead of a port number, comment out the `PortNumber` property and change `HostName` to `HostName=sqlserver.domain.company.com\named_instance`

   If your SQL server host uses Integrated Windows Authentication, uncomment and configure the final 2 lines in the example.
2. If you have not already done so, join the SQL server host machine to the Windows Active Directory domain.
3. Create a sample message flow that connects to the database and set up the **mqsisetdbparms** parameters and associated permissions in the SQL server as you would a normal ODBC connection. Now test that this initial setup works correctly.

What to do next

Now that these initial steps are complete, you must re-edit the odbc.ini as described in the next task.

Configuring IBM App Connect Enterprise to connect to SQL server with Kerberos: Part 2

The following task demonstrates the next steps in enabling Kerberos support for SQL server.

Before you begin

You must complete the steps in the previous task: "Configuring IBM App Connect Enterprise to connect to SQL server: Part 1" on page 2645

About this task

Now that the initial steps are complete, you must re-edit the odbc.ini file, and begin that next set of steps:

Procedure

1. Update the previous entry that you made to the odbc.ini file with the following changes:

```
[TESTDB]
Driver=/opt/IBM/mqsi/11.0.0.n/server/ODBC/drivers/lib/UKsqls95.so
Description=DataDirect SQL Server Wire Protocol
Database=TESTDB
HostName=sqlserver.ad.domain.company.com
PortNumber=1433
AnsiNPW=1
LoginTimeout=0
QueryTimeout=0
AuthenticationMethod=4
GSSClient=libgssapi_krb5.so
```

   In this example:

   - *TESTDB* is the name of a database.
   - *sqlserver.ad.domain.company.com* is the address of your SQL server in the Active Directory domain.
   - *1433* is the port number of your SQL server.
   - *libgssapi_krb5.so* is an existing Kerberos implementation on the system that is present and available through the library path. On AIX, use the following format: *GSSClient=libgssapi_krb5.a(libgssapi_krb5.a.so)*

2. Update the Kerberos configuration file, krb5.conf, on the Linux or UNIX machine.

   You can typically find the file in /etc or /etc/krb5. Add the following entries to the configuration file:

```
[libdefaults]
  default_realm = AD.DOMAIN.COMPANY.COM
  default_tkt_enctypes = rc4-hmac
  default_tgs_enctypes = rc4-hmac

[realms]
  AD.DOMAIN.COMPANY.COM = {
    kdc = adserver.ad.domain.company.com:88
    default_domain = ad.domain.company.com
  }

[domain_realm]
  .domain.company.com = AD.DOMAIN.COMPANY.COM
```

   where the Kerberos realm is your Active Directory domain name, which you must specify in upper case. kdc=adserver.ad.domain.company.com is the name of your host that you are running the Key Distribution Center for the Kerberos realm.

3. Use the **kinit** *username* command, where *username* is the user name that will be used to connect to the SQL server, to test that the krb5.conf file is set up to acquire a Ticket Granting Ticket for the user name.

4. Ensure that a Service Principal Name (SPN) for the SQL Server service is registered in the Active Directory.

   On starting, the SQL Service database engine normally attempts to register an SPN but if the SPN has not already been registered, issue the **setspn** command on the Active Directory, as in the following example:

   ```
   setspn -S MSSQLSvc/sqlserver.ad.domain.company.com:1433 windowsDomain\accountName
   ```

   where `sqlserver.ad.domain.company.com:1433` is the address and port number of your SQL server.
5. Open SQL Server Management Studio, and grant SQL Server login permission to the domain user on the server and on the database that you specified in the `odbc.ini` file in a previous step.
6. Run the following commands in a broker console on the Linux or UNIX client machine:

   ```
   mqsisetdbparms ACENODE -n odbc::TESTDB -u kerberos::username@AD.DOMAIN.COMPANY.COM -p
    N0chang3
   mqsichangeproperties ACENODE -e integration_server -o ComIbmJVMMangaer -n kerberosConfigFile
    -v /location_of_krb_config_file/krb.conf
   ```

   where *username* is the domain user, and *AD.DOMAIN.COMPANY.COM* is the address of your Kerberos realm.
7. Run the flow from "Configuring IBM App Connect Enterprise to connect to SQL server: Part 1" on page 2645 again. The broker now connects to SQL Server using the Kerberos ticket.

What to do next

Now that these steps are complete, you must continue to the next task as an administrator on your SQL Server machine.

Configuring IBM App Connect Enterprise to connect to SQL Server with Kerberos and SSL support: Part 3

The following task demonstrates the next steps in enabling SSL support for SQL server.

Before you begin

You must complete the steps in the previous tasks: "Configuring IBM App Connect Enterprise to connect to SQL server: Part 1" on page 2645 and "Configuring IBM App Connect Enterprise to connect to SQL server with Kerberos: Part 2" on page 2646. You must have administrator privileges on your SQL server machine.

About this task

Now that the initial steps are complete, you must log in as an administrator on your SQL Server machine to complete that next set of steps.

The following set of certificates are needed:

1. A certificate for the SQL Server.
2. A certificate for IBM App Connect Enterprise

Note: These certificates cannot be self-signed certificates, they must be issued by a trusted authority. Create a temporary certificate authority for development purposes if necessary, using openssl or lightweight CA software. You must ensure that the certificates match the machine names.

Complete the following steps.

Procedure

1. Open the Microsoft Management Console as an administrator on your SQL server machine.
2. Go to **File** > **Add/Remove snap in** and select **Certificates** > **Computer Account** > **Local Computer**.

3. Optional: (Complete this step only if you are using a local certificate authority). Expand the **Trusted Root Certification Authorities** folder.
4. Optional: (Complete this step only if you are using a local certificate authority). Right-click the open folder and select **All Tasks** > **import**. Browse and select your certificate authority root certificate `cacert.pem`, and import it to `Trusted Root Certification Authorities`.
5. Expand the **Personal** folder.
6. Right-click the open folder and select **All Tasks** > **import**. Browse and select the private key and certificate for the server as provided by the certificate authority , and import it to `Personal`.
7. Right-click on the newly imported *`sqlserver.domain.company.com`* certificate, and select **All Tasks** > **Manage Private Keys**
8. Open the SQL Server Configuration Manager, and select **SQL Server Services**.
9. Double-click the **SQL Server** entry and copy the **`Account Name`**.
10. Switch back to the Microsoft Management Console **Certificates** dialog and complete the following steps:and add the **`Account Name`** that you copied in the previous step, and search on the local machine.
    a) Click **Add**.
    b) Paste the **`Account Name`** that you copied in step "9" on page 2648
    c) Click **Check Names**.
    d) Click **OK**.
    e) Enable the account **Full Control** on the private keys.
    f) Click **OK**.
11. Switch back to the SQL Server Configuration Manager, and expand **SQL Server and Network Configuration.**
12. Right-click **Protocols for *SQLServer*** where *SQLServer* is the name you have used throughout these tasks, and select **Properties**.
    Open the **Certificate** tab.
13. Select the imported certificate from the drop-down box.
14. Select the **Flags** tab, and set **Force Encryption** to `Yes`.
15. Restart the SQL Server from the **SQL Services** window.
16. Update the odbc `.ini` file again to add the following configuration:

```
EncryptionMethod=1
TrustStore=/path/to/certificates/cacert.pem
```

17. Rerun the test application.

Security for the IBM App Connect Enterprise Toolkit

What to consider when you set up security for the IBM App Connect Enterprise Toolkit.

If administration security is enabled for a local integration node, the user ID that is used to run the IBM App Connect Enterprise Toolkit is passed to the integration node and used for authorization. For a remote integration node, the connection must be made using a web user ID, which is assigned to a role. For more information, see "Managing web user accounts" on page 2510.

If a user ID is a member of the group mqbrkrs, it has full permissions to act on all resources; for more information, see "Permissions for administrators" on page 2522. For information about authorizations for administration applications, see Tasks and authorizations for administration security.

Routing requests through an HTTP proxy server that has authentication enabled

You can send requests through an HTTP proxy server that requires authentication, by using the **mqsisetdbparms** command to specify credentials on the `httpproxy::` resource name.

About this task

You can configure the following message flow nodes to route requests through an HTTP proxy server, by setting the **HTTP(S) proxy location** parameter:

- HTTPAsyncRequest
- HTTPRequest
- RESTAsyncRequest
- RESTRequest
- SalesforceRequest
- SOAPAsyncRequest
- SOAPRequest

If the HTTP proxy server requires authentication, you can provide the user name and password by specifying the `httpproxy::` resource name through the **mqsisetdbparms** command.

Note: A message flow can provide the proxy authentication user name and password through the `ProxyConnectHeaders` property in the local environment for HTTPS requests, or the `Proxy-Authorization` header for HTTP requests. If these are present in the flow, they take precedence over values set by the **mqsisetdbparms** command; this applies to the HTTP, REST, and SOAP nodes listed above, but not to the SalesforceRequest node.

Procedure

Follow these steps to enable a message flow node to route requests through an HTTP proxy server:

1. Specify the URL of the proxy server in the **HTTP(S) proxy location** parameter of the node.
2. If the HTTP proxy server requires authentication through a user name and password, use the **mqsisetdbparms** command to provide these credentials on the `httpproxy::` resource name.

   You can set the resource name to apply to either an HTTP proxy with a specified host name, in the form `httpproxy::`*`proxy_hostname`*, or to any HTTP proxy, by specifying `httpproxy::HTTPPROXY`. If you set the resource name to apply to specific HTTP proxy server, the *proxy_hostname* must match the hostname of the configured proxy, without the port.

   For example, use the following command to associate a user name and password with a connection to a specified HTTP proxy server (in this example, *myProxyHostname*):

   ```
   mqsisetdbparms -w c:\workdir\ACEServ1 -n httpproxy::myProxyHostname -u myUserID -p myPassword
   ```

   Alternatively, use the following command to specify the credentials to be used as a default, for connecting to any HTTP proxy server that does not have a matching host name:

   ```
   mqsisetdbparms -w c:\workdir\ACEServ1 -n httpproxy::HTTPPROXY -u myProxyUsername -p
    myProxyPassword
   ```

3. If you add or modify the credentials by using the **mqsisetdbparms** command, you must restart the integration server for the changes to take effect.

   For more information, see command.

WS-Security

Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. WS-Security mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security is a message-level standard that is based on securing SOAP messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens. The web services security specification defines the facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. No specific type of security token is required by WS-Security. It is designed to be extensible, for example, to support multiple security token formats.

WS-Security also describes how to encode binary security tokens and attach them to SOAP messages. Specifically, the WS-Security profile specifications describe how to encode the following tokens:

- Username tokens
- X.509 certificates
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

With WS-Security, the domain of these mechanisms can be extended by carrying authentication information in web services requests. WS-Security also includes extensibility mechanisms that can be used to further describe the credentials that are included with a message. WS-Security is a building block that can be used in conjunction with other web service protocols to address a wide variety of application security requirements.

There are numerous advantages to using WS-Security.

- Different parts of a message can be secured in a variety of ways. For example, you can use integrity on the security token (user ID and password) and confidentiality on the SOAP message body.
- Intermediaries can be used and end-to-end message-level security can be provided through any number of intermediaries.
- WS-Security works across multiple transports and is independent of the underlying transport protocol.
- Authentication of both individual users and multiple party identities is possible.

Traditional Web security mechanisms, such as HTTPS, might be insufficient to manage the security requirements of all web service scenarios. For example, when an application sends a SOAP message using HTTPS, the message is secured only for the HTTPS connection, meaning during the transport of the message between the service requester (the client) and the service. However, the application might require that the message data be secured beyond the HTTPS connection, or even beyond the transport layer. By securing web services at the message level, message-level security is capable of meeting these expanded requirements.

Message-level security, or securing web services at the message level, addresses the same security requirements as for traditional Web security. These security requirements include: identity, authentication, authorization, integrity, confidentiality, nonrepudiation, and basic message exchange. Both traditional Web and message-level security share many of the same mechanisms for handling security, including digital certificates, encryption, and digital signatures.

With message-level security, the SOAP message itself either contains the information needed to secure the message or it contains information about where to get that information to handle security needs. The SOAP message also contains information relevant to the protocols and procedures for processing the specified message-level security. However, message-level security is not tied to any particular transport mechanism. Because the security information is part of the message, it is independent of a transport protocol, such as HTTPS.

The client adds to the SOAP message header security information that applies to that particular message. When the message is received, the web service endpoint, using the security information in the header, verifies the secured message and validates it against the policy. For example, the service endpoint might verify the message signature and check that the message has not been tampered with. It is possible to add signature and encryption information to the SOAP message headers, as well as other information such as security tokens for identity (for example, an X.509 certificate) that are bound to the SOAP message content.

Without message-level security, the SOAP message is sent in clear text, and personal information such as a user ID or an account number is not protected. Without applying message-level security, there is only a SOAP body under the SOAP envelope in the SOAP message. By applying features from the WS-Security specification, the SOAP security header is inserted under the SOAP envelope in the SOAP message when the SOAP body is signed and encrypted.

To keep the integrity or confidentiality of the message, digital signatures and encryption are typically applied.

- Confidentiality specifies the confidentiality constraints that are applied to generated messages. This includes specifying which message parts within the generated message must be encrypted, and the message parts to attach encrypted Nonce and time stamp elements to.
- Integrity is provided by applying a digital signature to a SOAP message. Confidentiality is applied by SOAP message encryption.

You can add an authentication mechanism by inserting various types of security tokens, such as the Username token (element). When the Username token is received by the web service server, the user name and password are extracted and verified. Only when the user name and password combination is valid, will the message be accepted and processed at the server. Using the Username token is just one of the ways of implementing authentication. This mechanism is also known as basic authentication.

The OASIS Web Services Security Specification provides a set of mechanisms to help developers of web services secure SOAP message exchanges. For details of the OASIS Web Services Security Specification, see OASIS Standard for WS-Security Specification.

WS-Security mechanisms

The WS-Security specification provides three mechanisms for securing web services at the message level: authentication, integrity, and confidentiality.

Authentication

This mechanism uses a security token to validate the user and determine whether a client is valid in a particular context. A client can be a user, computer, or application. Without authentication, an attacker can use spoofing techniques to send a modified SOAP message to the service provider.

In authentication, a security token is inserted in the request message. Depending on the type of security token that is being used, the security token can also be inserted in the response message. The following types of security token are supported for authentication:

- Username tokens
- X.509 certificates
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

Username tokens are used to validate user names and passwords. When a web service server receives a username token, the user name and password are extracted and passed to a user registry for verification. If the user name and password combination is valid, the result is returned to the server and the message is accepted and processed. When used in authentication, username tokens are typically passed only in the request message, not the response message.

X.509 tokens are validated by using a certificate path.

The integration node support for SAML assertions is restricted to passing the token to a WS-Trust Security Token Service (STS) for validation.

Kerberos tickets are validated against the host's Kerberos `keytab` file.

The integration node support for LTPA binary tokens is restricted to passing the token to a WS-Trust STS for validation.

All types of token must be protected. For this reason, if you send them over an untrusted network, take one of the following precautions:

- Use HTTPS
- Configure the policy set to protect the appropriate elements in the SOAP header

Integrity

This mechanism uses message signing to ensure that information is not changed, altered, or lost accidentally. When integrity is implemented, an XML digital signature is generated on the contents of a SOAP message. If unauthorized changes are made to the message data, the signature is not validated. Without integrity, an attacker can use tampering techniques to intercept a SOAP message between the web service client and server, and modify it.

Confidentiality

This mechanism uses message encryption to ensure that no party or process can access or disclose the information in the message. When a SOAP message is encrypted, only a service that knows the appropriate key can decrypt and read the message.

Implementing WS-Security

Configure authentication, XML encryption, XML signature, and message expiration by using the WS Policy Sets and Bindings editor.

About this task

You use the WS Policy Sets and Bindings editor in the IBM App Connect Enterprise Toolkit to configure the following aspects of WS-Security:

"Authentication" on page 2652
"Confidentiality" on page 2654
"Integrity" on page 2655
"Expiration" on page 2655

Authentication

About this task

The following tokens are supported:

- Username
- X.509
- SAML assertions
- Kerberos tickets
- LTPA binary tokens

Configuring authentication with username tokens:

1. In the IBM App Connect Enterprise Toolkit, create a policy project by clicking **File** > **New** > **Policy Project**.

2. In the Application Development view, right-click the policy project, then click **New** > **WS Policy Sets and Bindings**. The WS Policy Sets and Bindings editor opens.
3. Click **Add** to create a policy set.
4. Select the new policy in the tree view, then click **Add WS-Security** to add the policy type to the policy set.
5. Expand the WS-Security policy type in the tree view, click **Authentication Tokens**, then click **Add** to add UserName authentication tokens to the policy (see Policy Sets and Policy Set Bindings editor: Authentication tokens panel).
6. Repeat the previous step to configure X.509 authentication tokens (see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel).
7. Configure a security profile (see “Message flow security and security profiles” on page 2673).
8. Associate the policy set with a message flow or node (see “Associating policy sets and bindings with message flows and nodes” on page 2672).

Configuring authentication with X.509 tokens:

1. If you are using the integration node's truststore to hold the trusted certificate, you must configure it. See “Viewing and setting keystore and truststore runtime properties at integration node level” on page 2667 or “Viewing and setting keystore and truststore runtime properties at integration server level ” on page 2669, depending on where you want to set keystore and truststore runtime properties.
2. Create a policy set (as described in the previous section), then add UserName and X.509 authentication tokens to it.
3. Configure the certificate mode for either integration node truststore or an external security provider (see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel).
4. If you are using an external security provider, configure a security profile (see “Message flow security and security profiles” on page 2673).
5. Associate the policy set with a message flow or node (see “Associating policy sets and bindings with message flows and nodes” on page 2672).

Configuring authentication with SAML assertions:

1. Create a policy set and add SAML pass-through 1.1 or SAML pass-through 2.0 tokens to it (see Policy Sets and Policy Set Bindings editor: Authentication tokens panel). SAML pass-though does not enforce subject confirmation, but the assertion is simply provided as a token to be processed in the external Security Token Server specified in the security profile that is associated with the node.
2. Configure a security profile. The security profile must be configured to use a WS-Trust v1.3 STS (see “Message flow security and security profiles” on page 2673).
3. Associate the policy set with a message flow or node (see “Associating policy sets and bindings with message flows and nodes” on page 2672).

Configuring authentication with Kerberos tickets:

1. Create a policy set and add your Kerberos token type as symmetric tokens (see Policy Sets and Policy Set Bindings editor: Message Level Protection panel).
2. Associate the policy set with a message flow or node (see “Associating policy sets and bindings with message flows and nodes” on page 2672).
3. Configure the host's Kerberos `keytab` file. For more information about Kerberos configuration, see the documentation for your integration node's host system. For example, for Windows, see the "Step-by-Step Guide to Kerberos 5 (krb5 1.0) Interoperability", which you can access at `http://technet.microsoft.com/en-us/library/`.

Configuring authentication with LTPA binary tokens:

1. Create a policy set and add LTPA tokens to it (see Policy Sets and Policy Set Bindings editor: Authentication tokens panel). The LTPA binary token is passed through to the external Security Token Service (STS) specified in the security profile that is associated with the node.
2. Configure a security profile. The security profile must be configured to use a WS-Trust v1.3 STS (see "Message flow security and security profiles" on page 2673).
3. Associate the policy set with a message flow or node (see "Associating policy sets and bindings with message flows and nodes" on page 2672).

Confidentiality

About this task

Confidentiality is provided by XML encryption, and requires either X.509 tokens or Kerberos tickets.

Configuring XML encryption with X.509 tokens:

1. If you are using the integration node's truststore to hold the trusted certificate, you must configure it. See "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667 or "Viewing and setting keystore and truststore runtime properties at integration server level " on page 2669, depending on where you want to set keystore and truststore runtime properties.
2. Create a policy set, enable XML encryption, create encryption tokens, and select the encryption algorithms that you will use (see Policy Sets and Policy Set Bindings editor: Message Level Protection panel).
3. Define which parts of a message are to be encrypted (see Policy Sets and Policy Set Bindings editor: Message Part Protection panel).
4. Further configure message part encryption (see Policy Sets and Policy Set Bindings editor: Message Part Policies panel).
5. Further configure the keystore and truststore (see Policy Sets and Policy Set Bindings editor: Key Information panel).
6. Associate the policy set with a message flow or node (see "Associating policy sets and bindings with message flows and nodes" on page 2672).

Configuring XML encryption with Kerberos tickets:

1. Configure your host for Kerberos, providing a `krb.conf` configuration file. This step is required on all operating systems, including Windows.
2. Provide the integration node with the Kerberos client credentials for accessing the Kerberos Key Distribution Center (KDC). These credentials (which are required for SOAPRequest nodes) can be provided in the Integration node properties tree, or by using the **mqsisetdbparms** command. The credentials are taken in order of priority:
   - The node has a security profile with the `Propagation` property set to *True* and the Properties tree UserName and password token is present. If no UserName and password token exists, an exception in thrown.
   - ```
     mqsisetdbparms kerberos::<realm>::<integrationServerName>
     ```
   - ```
     mqsisetdbparms kerberos::<realm>
     ```
   - ```
     mqsisetdbparms kerberos::kerberos
     ```
3. Create a policy set and add the required Kerberos token type as Symmetric Tokens (see Policy Sets and Policy Set Bindings editor: Message Level Protection panel).
     ```

Integrity

About this task

Integrity is provided by XML signature, and requires either X.509 tokens or Kerberos tickets.

Configuring XML signature with X.509 tokens:

1. If you are using the integration node's truststore to hold the trusted certificate, you must configure it. See “Viewing and setting keystore and truststore runtime properties at integration node level” on page 2667 or “Viewing and setting keystore and truststore runtime properties at integration server level ” on page 2669, depending on where you want to set keystore and truststore runtime properties.
2. Create a policy set, enable XML signature, and create signature tokens (see Policy Sets and Policy Set Bindings editor: Message Level Protection panel).
3. Define which parts of a message are to be signed (see Policy Sets and Policy Set Bindings editor: Message Part Protection panel).
4. Further configure message part signature (see Policy Sets and Policy Set Bindings editor: Message Part Policies panel).
5. Further configure the keystore and truststore (see Policy Sets and Policy Set Bindings editor: Key Information panel).
6. Associate the policy set with a message flow or node (see “Associating policy sets and bindings with message flows and nodes” on page 2672).

Configuring XML signature with Kerberos tickets:

1. Configure your host for Kerberos, providing a `krb.conf` configuration file. This step is required on all operating systems, including Windows.
2. Provide the integration node with the Kerberos client credentials for accessing the Kerberos Key Distribution Center (KDC). These credentials (which are required for SOAPRequest nodes) can be provided in the Integration node properties tree, or by using the **mqsisetdbparms** command. The credentials are taken in the following order of priority:
   - The node has a security profile with the `Propagation` property set to *True* and the Properties tree UserName and password token is present. If no UserName and password token exists, an exception in thrown.
   - ```
     mqsisetdbparms kerberos::<realm>::<integrationServerName>
     ```
   - ```
     mqsisetdbparms kerberos::<realm>
     ```
   - ```
     mqsisetdbparms kerberos::kerberos
     ```
3. Create a policy set and add the required Kerberos token type as Symmetric Tokens (see Policy Sets and Policy Set Bindings editor: Message Level Protection panel).

Expiration

Procedure

To configure message expiration, see Policy Sets and Policy Set Bindings editor: Message Expiration panel.

Kerberos-based WS-Security

You can use Kerberos authentication with WS-Security either as a service or as a client.

Kerberos is a network authentication protocol that enables mutual authentication with symmetric keys. Users and services on a network authenticate with each other through a Key Distribution Center (KDC), as
     ```

a trusted third party. IBM App Connect Enterprise provides support for Kerberos either as a service or as a client.

You can use message flows to call web services that are secured with Kerberos by using a SOAP Request node. You can also provide web services that are secured with Kerberos by using SOAP Input Nodes. The WS-Security header passes Kerberos tokens. You can sign and encrypt either parts or all of a SOAP message by using Kerberos tokens. Signing and encrypting messages provides message integrity, confidentiality, and authenticity.

For information about Kerberos terminology and concepts, see Concepts for Kerberos security.

You can configure IBM App Connect Enterprise to act as a Kerberos secured service or as a client to a Kerberos secured service. Install a Kerberos KDC and configure it to contain all the principals that are required by the environment, then configure for a service or client. For more information about configuring Kerberos, see your host Kerberos documentation.

- For the steps needed to embed IBM App Connect Enterprise as a client, see Configuring IBM App Connect Enterprise as a client to a Kerberos secured Service.
- For the steps needed to configure IBM App Connect Enterprise as a secured service, see Configuring IBM App Connect Enterprise as a Kerberos secured Service.
- For the steps needed to configure separate Kerberos configuration files for each integration server, see “Configuring separate Kerberos configuration files for each integration server” on page 2660.

Kerberos security concepts

Learn about Kerberos authentication and WS-Security concepts.

Concepts

Kerberos configuration file
The configuration file contains information that is important for authentication and access. The configuration file contains the key distribution center (KDC) realm and location, supported encryption types, and keytab file location. Clients use the configuration file to authenticate with the KDC and request access to networked services. Kerberos secured services also use the configuration file to locate the keytab file that contains the private key that is associated with it.

Kerberos keytab file
The keytab file contains the principal and encrypted private key that is associated with the principal. The keytab file is created by exporting a principal from the KDC. Using the keytab file, a service can check the authenticity of a client and provide authentication without contacting the KDC.

Key Distribution Center (KDC)
The Key Distribution Center stores user and service principals with their associated key. A combination of either a user name and a password or a service name and a password provide the key. The KDC also provides an authentication server and a server that grants tickets.

Principals
Networked users and Services are known as principals. They authenticate with a Key Distribution Center (KDC).

Realm
The unique range of control that is provided by the KDC. By convention the realm is the DNS domain name that is converted to uppercase.

Service Principal Name (SPN)
The service principal name represents a unique, networked service. For a client to use a Kerberos secured service, the client must authenticate with a KDC and provide the SPN for the service. The ticketing server provides a ticket to the client that allows it to authenticate itself to the service.

Configuring IBM App Connect Enterprise as a client to a Kerberos secured service

You can configure IBM App Connect Enterprise to operate as a client to a Kerberos secured service for message integrity, confidentiality, and authenticity.

Before you begin

You must have access to a Key Distribution Center (KDC) and a server that is hosting the service. For more information about configuring Kerberos, see your host Kerberos documentation.

Procedure

1. Set the user credentials that are used to authenticate with the KDC.

   - You can configure the credentials at the integration node level by issuing a **mqsisetdbparms**. For example,

     ```
     mqsisetdbparms integrationNodeName -n SPN::realm -u username -p password
     ```

   - You can also set the user credentials at the integration server level. For example, you can set a specific realm in any integration server with

     ```
     mqsisetdbparms integrationNodeName -n kerberos::realm1::integrationServerName -u clientId -
     p password
     ```

   - You can also use the Properties tree to set the credentials by using the following ESQL in a compute node:

     ```
     SET OutputRoot.Properties.IdentitySourceType = 'usernameAndPassword';
     SET OutputRoot.Properties.IdentitySourceToken = Username;
     SET OutputRoot.Properties.IdentitySourcePassword = Password;
     ```

2. Create a Kerberos configuration file. The client can authenticate with the KDC, using the configuration file.

   For more information about Kerberos-based WS-Security that is supported in SOAP nodes, see Message flow security and security profiles.

   When you use Kerberos for security, the default Kerberos configuration file is the one on your workstation. The location for the configuration file differs depending on the system. The usual locations are:

   - For Windows - `C:\Windows\krb5.ini` and `C:\WINNT\krb5.ini`
   - For Linux - `/etc/krb5.conf` , UNIX (AIX) `/etc/krb5/krb5.conf`
   - For z/OS - `/krb5/krb5.conf`

   You can configure Kerberos configuration files for use by an integration node or integration server.

   The following sample Kerberos configuration file shows typical values for the variables. The variables *default_realm*, *default_keytab_name*, and the names in the *realms* are among the values you change in the configuration file, depending on your network and location of the configuration file.

   ```
   [libdefaults]
   default_realm = MYREALM.EXAMPLE.COM
   default_keytab_name = FILE:c:\Windows\krb5.keytab
   default_tkt_enctypes = rc4-hmac
   default_tgs_enctypes = rc4-hmac
   dns_lookup_realm = false
   dns_lookup_kdc = false
   ticket_lifetime = 24h
   renew_lifetime = 7d
   forwardable = true
   [realms]
   MYREALM.EXAMPLE.COM = {
   kdc = kdc.myrealm.example.com
   admin_server = kdc.myrealm.example.com
   ```

```
}
```

For example, you can set the variables for an IBM App Connect Enterprise level Kerberos configuration with

```
mqsichangeproperties integrationNodeName -o BrokerRegistry -n brokerKerberosConfigFile -
v kerberosConfigLocation
```

For example, you can set the variables for an integration server level Kerberos configuration with

```
mqsichangeproperties integrationNodeName -e integrationServerName -o ComIbmJVMManager -n
 brokerKerberosConfigFile -v kerberosConfigLocation
```

3. Configure a policy set and binding that is associated with the SOAPRequest node for the BAR containing the message flow.
   - To configure the Kerberos token with a policy set, see Policy Sets and Policy Set Bindings editor: Message Level Protection panel.
   - To configure a consumer message part policy, see Policy Sets and Policy Set Bindings editor: Message Part Protection panel.
   - To optionally sign or encrypt message parts using the policy set, see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.
   - To configure Kerberos specific settings such as target service name and target service realm, see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.

Results

You have configured IBM App Connect Enterprise to operate as a client to a Kerberos secured service.

Configuring IBM App Connect Enterprise as a Kerberos secured service

You can configure IBM App Connect Enterprise to operate as a Kerberos secured service for message integrity, confidentiality, and authenticity.

Before you begin

You must have access to a Key Distribution Center (KDC) and a server that is hosting the Kerberos secured service. For more information about configuring Kerberos, see your host Kerberos documentation.

About this task

Use this task to configure Kerberos as a secured service for IBM App Connect Enterprise.

Procedure

1. Export a keytab that contains the private key of the service principal from the KDC.
   For example:

```
ktpass -out c:\Windows\krb5.keytab -princ SomePrincipal@YourDomain
```

```
  -crypto RC4-HMAC-NT mapUser Username -pass Password -mapOp set
```

where

out *filename*
Specifies the name and path of the keytab file to be generated.

princ *principal_name*
Specifies the principal name.

crypto *encryption_type*
Specifies the encryption type.

mapUser *username*
Maps the name of a Kerberos principal to a local account.

pass *password*
Specifies the password to use for this principal name.

mapOp *attribute*
Defines how the mapping attribute is set. The attribute alternatives are either `add` or `set`.

2. Copy the keytab file to the server that hosts the service. You can copy the file to the server by exporting the keytab file and transferring it to the server, for instance by using FTP. The Kerberos configuration file contains a reference to the keytab file in the form of a file URL (such as: `/home/user/my.keytab`).

   Because the reference is in the configuration file on the server, the server service can take on the Kerberos principal that is defined in the keytab.

3. Create a Kerberos configuration file that specifies the location of the keytab file on the local workstation.

   You can use more than one service principal name per integration node per Kerberos realm. Use your workstation default Kerberos configuration file when you are using Kerberos for security. The location for the configuration file differs depending on the system. The usual locations are:

   - Windows: `C:\Windows\krb5.ini` and `C:\WINNT\krb5.ini`
   - Linux : `/etc/krb5.conf`
   - UNIX (AIX): `/etc/krb5/krb5.conf`
   - z/OS: `/krb5/krb5.conf`

   Different Kerberos configuration files can be configured for use by the integration node and integration servers.

   The following sample Kerberos configuration file shows typical values for the variables. The variables *default_realm*, *default_keytab_name*, and the names in the *realms* are among the values you change in the configuration file, depending on your network and location of the configuration file.

```
[libdefaults]
default_realm = MYREALM.EXAMPLE.COM
default_keytab_name = FILE:c:\Windows\krb5.keytab
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
[realms]
MYREALM.EXAMPLE.COM = {
kdc = kdc.myrealm.example.com
admin_server = kdc.myrealm.example.com
}
```

4. Create a new Kerberos configuration file to use unique Kerberos Service Principals per integration node or per integration server. You do so by specifying the keytab file that contains the required service principal.

5. Use one of these **mqsichangeproperties** commands to specify the location of your new configuration file.
   - For an integration node level Kerberos configuration:

     ```
     mqsichangeproperties integrationNodeName -o BrokerRegistry
     -n brokerKerberosConfigFile -v kerberosConfigLocation
     ```

   - For an integration server level Kerberos configuration:

     ```
     mqsichangeproperties integrationNodeName -e integrationServerName
     -o ComIbmJVMManager -n brokerKerberosConfigFile -v kerberosConfigLocation
     ```

6. Configure a policy set and binding that is associated with the SOAPInput node for the BAR containing the message flow.
   - To configure the Kerberos token with a policy set, see Policy Sets and Policy Set Bindings editor: Message Level Protection panel.
   - To configure a provider message part policy, see Policy Sets and Policy Set Bindings editor: Message Part Protection panel.
   - To sign or encrypt message parts optionally by using a policy set, see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.
   - To configure Kerberos specific settings (such as, target service name and target service realm), see Policy Sets and Policy Set Bindings editor: Message Part Policies panel.

Results

You configured IBM App Connect Enterprise to be a Kerberos secured service.

Configuring separate Kerberos configuration files for each integration server

You can create multiple Kerberos configuration files and then use a separate Kerberos configuration file with each integration server.

Before you begin

You must have access to a Key Distribution Center (KDC) and a server that is hosting the Kerberos-secured service, or you must have an existing Kerberos configuration file. For more information about configuring Kerberos, see your host Kerberos documentation.

About this task

Each integration server in an integration node has a separate JVM, and each JVM has a configuration setting for a Kerberos configuration file. Complete the following steps to configure a separate Kerberos configuration file for each integration server:

Procedure

1. Create a Kerberos configuration file or identify an existing Kerberos configuration file.

   When you use Kerberos for security, the default Kerberos configuration file is the one on your workstation. The location for the configuration file differs depending on the system. The usual locations are as follows:

   - For Windows: `C:\Windows\krb5.ini` or `C:\WINNT\krb5.ini`
   - For Linux: `/etc/krb5.conf`
   - For UNIX (AIX): `/etc/krb5/krb5.conf`
   - For z/OS: `/krb5/krb5.conf`

2. Create a new directory for each integration server.
   For example, on Windows, you might create the following directories:
   - *work_path*/krb5/server1 (to hold the copy of the Kerberos configuration file for integration server "server1")
   - *work_path*/krb5/server2 (to hold the copy of the Kerberos configuration file for integration server "server2")

   where *work_path* is the machine-wide IBM App Connect Enterprise working directory. By using this directory, you can manage the configuration of both the active and the standby integration nodes in a high availability deployment.

   Note: To verify the machine-wide IBM App Connect Enterprise working directory, enter the following command in a command console:

   ```
   echo %MQSI_WORKPATH%
   ```
3. Add a copy of the default Kerberos configuration file to each of the new directories.
4. If you have an integration server that exposes one or more services, generate a keytab file for each service and place it in the directory for that integration server. For information about creating keytab files, see " Configuring IBM App Connect Enterprise as a Kerberos secured service" on page 2658.
5. Edit each of the Kerberos configuration files and add the path to the keytab file (if required) and the realm information.
   For example, on Windows, you might add the following entries:

   ```
   default_keytab_name=work_path/krb5/server1/krb.keytab
   default_realm=server1Realm
   ```

   where *server1Realm* is the realm name for the integration server "server1". If you have to support multiple realms, you can also add entries for each realm into the [realms] section. The following code shows an example of the configuration for multiple realms:

   ```
   [libdefaults]
     default_keytab_name = FILE:/krb/test.keytab
     default_realm = REALM2.MYCO.COM
   #  default_realm = REALM1.MYCO.COM
     default_tkt_enctypes = rc4-hmac
     default_tgs_enctypes = rc4-hmac

   [realms]
     REALM1.MYCO.COM = {
       kdc = kdc1.myco.com:88
       admin_server = kdc.myco.ibm.com:749
       default_domain = myco.com
     }
     REALM2.MYCO.COM = {
       kdc = kdc2.myco.com:88
       default_domain = realm2.myco.com
     }

   [domain_realm]
   # defines how to map from internet domain names to Kerberos realms
     REALM2.MYCO.COM.MYCO.COM = REALM2.MYCO.COM
     .myco.com = REALM1.MYCO.COM
     .test.myco.com = REALM1.MYCO.COM
   ```
6. For each integration server, run the following command on your integration node:

   ```
   mqsichangeproperties node -e server -n jvmSystemProperty -v "-Djava.security.krb5.conf=path"
   ```

   where *node* is the name of your integration node, *server* is the name of your integration server, and *path* is the path to your Kerberos configuration file.
7. Remove the java.security.krb5.conf setting from **IBM_JAVA_OPTIONS** environment variable and restart your integration node.

Results

You configured a separate Kerberos configuration file (and optional keytab files) for each integration server.

Policy sets

Policy sets and bindings define and configure your WS-Security and WS-RM requirements, supported by IBM App Connect Enterprise, for the SOAPInput, SOAPReply, SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes.

A *policy set* is a container for the WS-Security and WS-RM policy types.

A policy set *binding* is associated with a policy set and contains information that is specific to the environment and platform, such as information about keys.

Use policy sets and bindings to define the following items for both request and response SOAP messages:

- Authentication for the following tokens:
  - Username tokens (requires a security profile to specify the external security provider)
  - X.509 certificates (requires the integration node keystore and truststore, or a security profile to specify the external security provider)
  - SAML assertions, using SAML 1.1 or 2.0 pass-through (requires a security profile to specify the external security provider)
  - LTPA tokens, using LTPA pass-through (requires a security profile to specify the external security provider)
- Asymmetric encryption (confidentiality) using X.509 certificates (requires the integration node keystore and truststore)
- Symmetric encryption (confidentiality) using Kerberos tokens (requires the host to be configured for Kerberos)
- Asymmetric signature (integrity) (requires the integration node keystore and truststore)

Either the whole SOAP message body, or specific parts of the SOAP message header and body can be encrypted and signed.

You can administer policy sets and bindings from the IBM App Connect Enterprise Toolkit, where you can add, delete, display and edit them. However, policy sets and bindings are non-dynamic, which means that they cannot be changed or redeployed while applications and message flows are using them. Before you can change and redeploy them, you must restart all applications on the integration server. You can do this by specifying the **`--restart-all-applications`** parameter on the mqsideploy or ibmint deploy command. Alternatively, if you try to redeploy policy sets and bindings by using the Toolkit, the initial attempt fails and you are then offered the option of restarting all applications and redeploying.

You can also export policy sets and bindings from an integration node (see "Exporting a policy set and policy set binding" on page 2673).

Policy sets are associated with a message flow, a message node, or both, in the BAR file editor. For convenience, you can specify settings for *provider* and *consumer* at the message flow level. The provider setting applies to all SOAPInput and SOAPReply nodes in the message flow. The consumer setting applies to all SOAPRequest, SOAPAsyncRequest, and SOAPAsyncResponse nodes. Individual policy set and binding assignments can be applied at the node level in the BAR editor, and these take precedence over the flow-level provider and consumer settings. The default setting is *none*, meaning that no policy set and bindings are to be used. For more information, see "Associating policy sets and bindings with message flows and nodes" on page 2672.

Several nodes in the same message flow can refer to the same policy set and bindings. It is the responsibility of the administrator to ensure that the required policy sets are available to the integration node at run time. An error is reported if the integration node cannot find the associated policy set or bindings.

The rest of this topic describes some of the terms that you will meet when configuring policy sets and bindings.

Default policy set and bindings

When an integration node is created, default policy set and bindings are created called `WSS10Default`. This default policy set contains a limited security policy, which specifies that a Username token is present in request messages (inbound) to SOAPInput nodes in the associated message flow. A default WS-RM policy set called `WSRMDefault` is also created.

The default policy set binding refers to the default `WSS10Default` policy set, but not to `WSRMDefault`. They are not editable.

Consumer and provider nodes

Nodes are either consumers or providers.

Consumer nodes

SOAPRequest
SOAPAsyncRequest
SOAPAsyncResponse

Provider nodes

SOAPInput
SOAPReply

Request and response

Request and response is a message exchange pattern (MEP). It describes a client that sends a SOAP Request message to a web services server, which in turn sends a Response SOAP message back to the client. The Request message is always the SOAP message from the client to the server, and the Response message is always the SOAP message reply from server to the client. The following table describes this pattern in relation to the IBM App Connect Enterprise SOAP nodes:

Node	Integration node viewpoint	Request	Response
SOAPInput	SOAP message inbound	Inbound message	Not applicable
SOAPReply	SOAP message outbound	Not applicable	Outbound message
SOAPRequest	SOAP message outbound followed by a SOAP message inbound	Outbound message	Inbound message
SOAPAsyncRequest	SOAP message outbound	Outbound message	Not applicable
SOAPAsyncResponse	SOAP message inbound	Not applicable	Inbound message

Initiator and recipient

Initiator and recipient are roles defined in the exchange of SOAP messages.

Initiator
The role that sends the initial message in a message exchange.

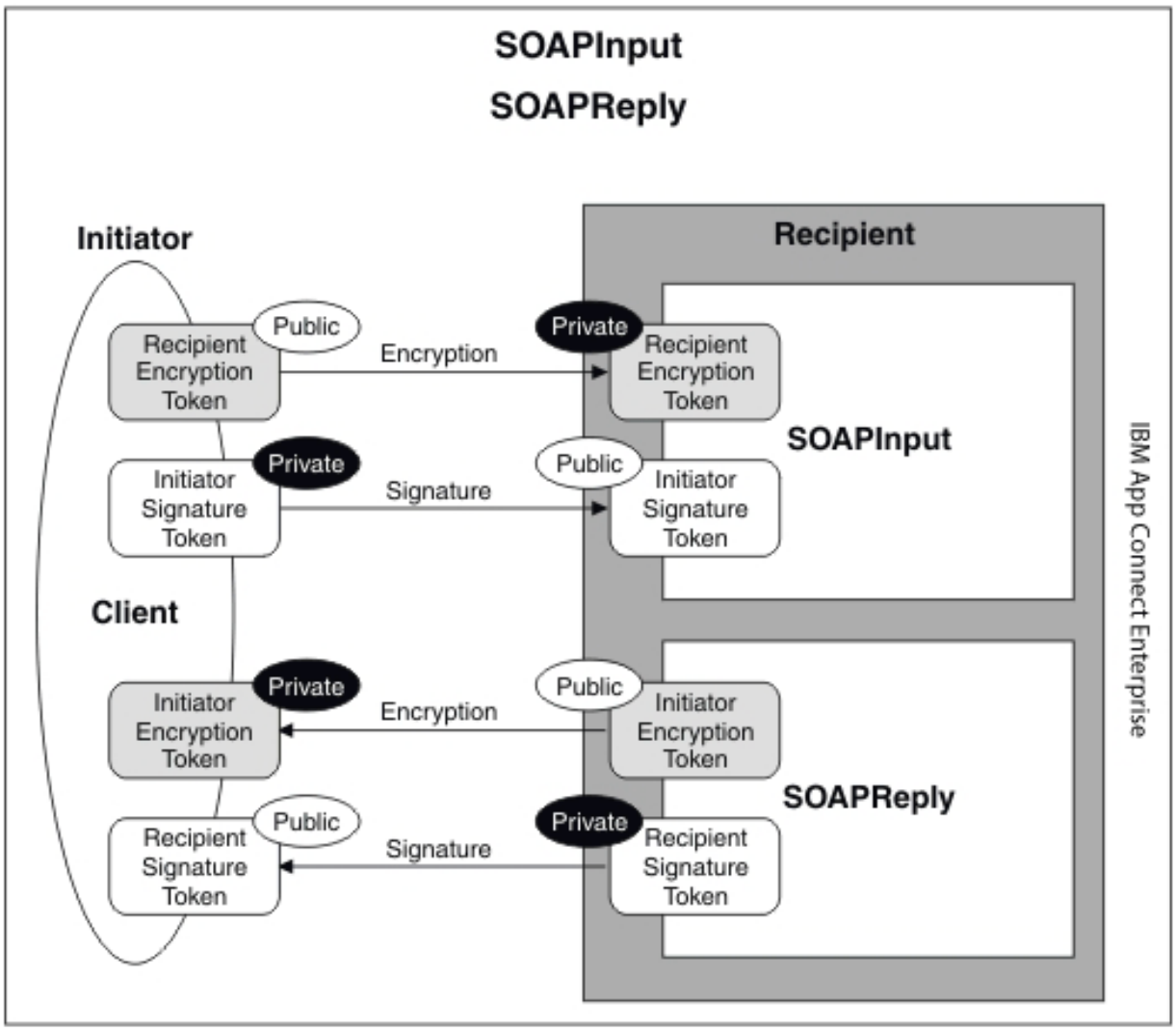
Recipient
The targeted role to process the initial message in a message exchange.

The following table describes these roles in relation to the SOAP nodes:

Node	Integration node viewpoint	Initiator	Recipient
SOAPInput	SOAP message inbound	External client sending SOAP message to the integration node.	SOAPInput node
SOAPReply	SOAP message outbound	External client that sent the original SOAP message to the integration node.	SOAPReply node
SOAPRequest (outbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPRequest (inbound)	SOAP message outbound followed by a SOAP message inbound	SOAPRequest node	External provider receiving the SOAP message
SOAPAsyncRequest	SOAP message outbound	SOAPAsyncRequest node	External provider receiving the SOAP message
SOAPAsyncResponse	SOAP message inbound	SOAPAsyncRequest node	External provider receiving the SOAP message

SOAPInput and SOAPReply nodes

In this diagram, the integration node acts as recipient. A SOAPInput node receives a message from a client (initiator). A SOAPReply node replies. Inbound and outbound messages are signed and encrypted.



In the request:

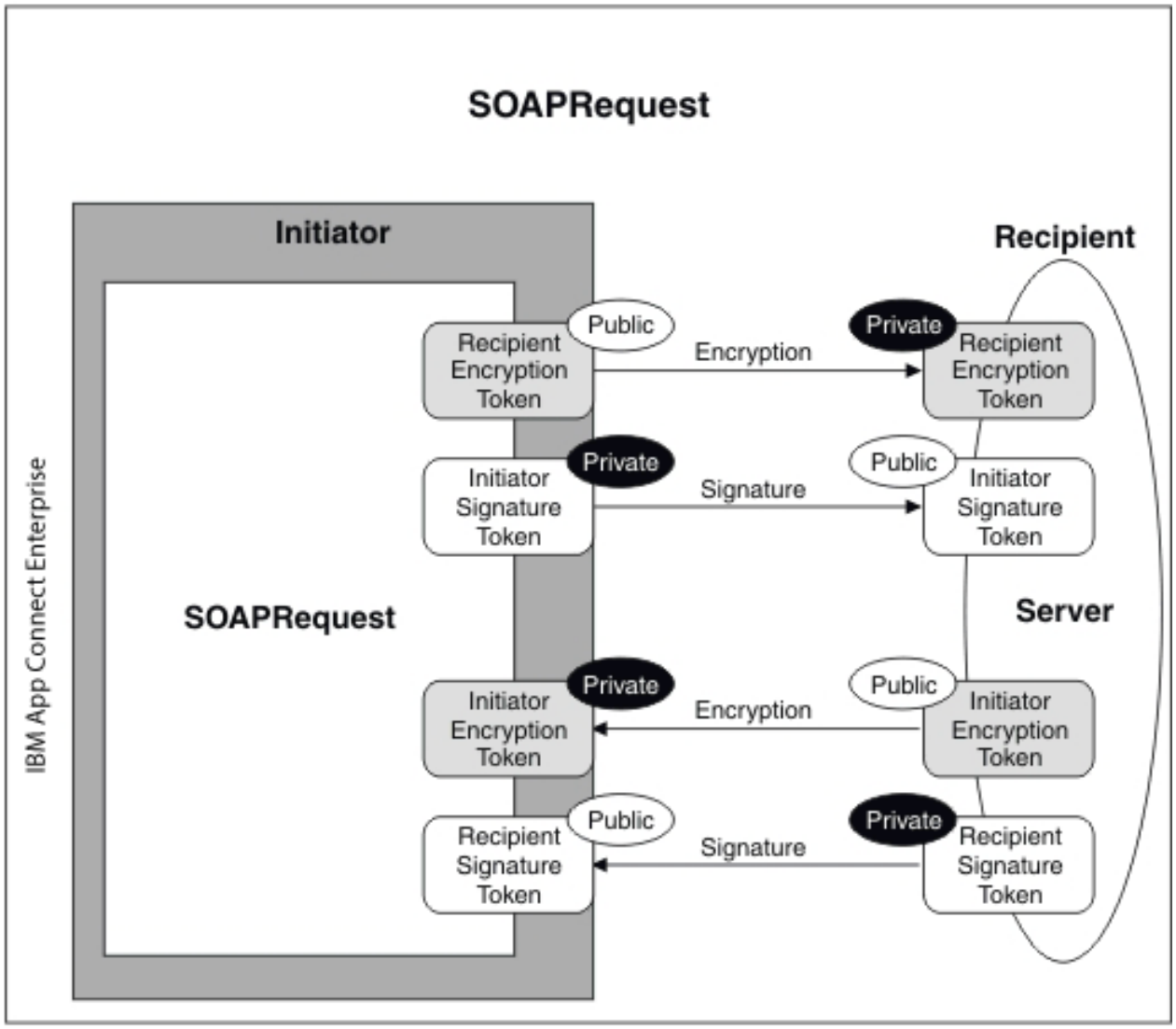
- The initiator uses the public encryption token of the integration node to encrypt the message, and its own private signature token to sign it.
- The integration node uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

In the response:

- The integration node uses the initiator's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The initiator uses its own private encryption token to decrypt the message, and the public signature token of the integration node to verify the signature.

SOAPRequest node

This diagram shows the integration node acting as an initiator. It uses the SOAPRequest node to make a synchronous request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted. Use of tokens is similar to the example of the asynchronous SOAP nodes, shown earlier.



In the request:

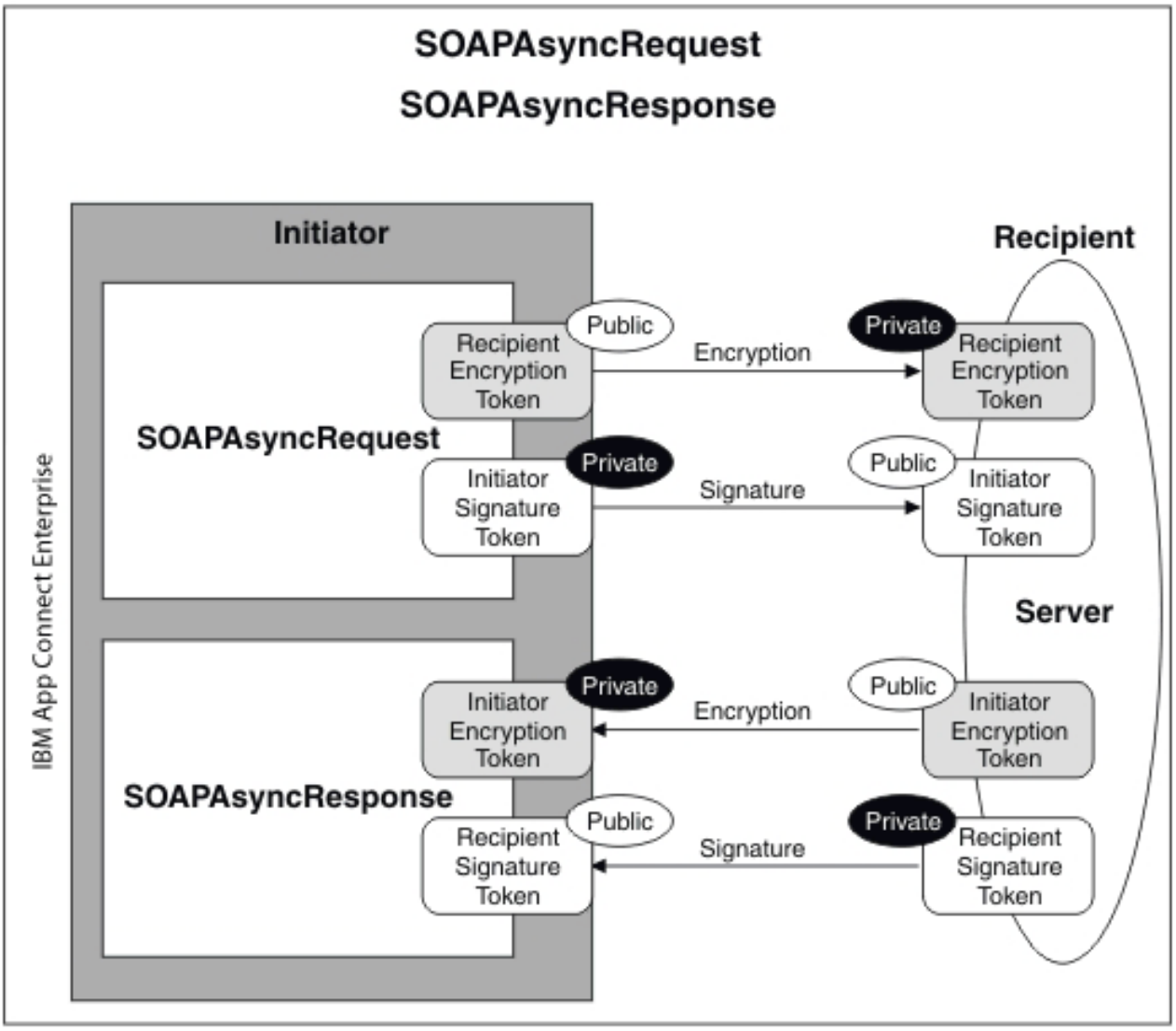
- The integration node uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the public signature token of the integration node to verify the signature.

In the response:

- The recipient uses the public encryption token of the integration node to encrypt the message, and its own private signature token to sign the message.
- The integration node uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Asynchronous SOAP nodes

This diagram shows the integration node acting as an initiator. It uses the asynchronous SOAP nodes to make a request to an external provider (the recipient). Inbound and outbound messages are signed and encrypted.



In the request:

- The integration node uses the recipient's public encryption token to encrypt the message, and its own private signature token to sign the message.
- The recipient uses its own private encryption token to decrypt the message, and the public signature token of the integration node to verify the signature.

In the response:

- The recipient uses the public encryption token of the integration node to encrypt the message, and its own private signature token to sign the message.
- The integration node uses its own private encryption token to decrypt the message, and the initiator's public signature token to verify the signature.

Viewing and setting keystore and truststore runtime properties at integration node level

Configure the integration node to refer to a keystore, a truststore, or both, before deploying any message flows that require policy set or bindings for signature, encryption, or X.509 Authentication.

About this task

Keystores and truststores are both keystores. They differ only in the way in which they are used.

- Put all private keys and public key certificates (PKC) in the keystore.
- Put all trusted root certificate authority (CA) certificates in the truststore. These certificates are used to establish the trust of any inbound public key certificates.

The only supported type of store is Java keystore (JKS).

Each instance of an integration node can be configured to refer to one keystore and one truststore.

The following properties of the integration node registry component must be defined correctly for policy sets and bindings:

brokerKeystoreFile
The directory and file location of the keystore.

brokerTruststoreFile
The directory and file location of the truststore.

To check what security properties you have set for an integration node, use the **mqsireportdbparms** command.

Listing existing integration node registry entries

About this task

To display all integration node registry values, run the following command:

```
mqsireportproperties integrationNodeName -o BrokerRegistry -a
```

This command returns entries like these:

```
BrokerRegistry=''
  uuid='BrokerRegistry'
  brokerKeystoreType='JKS'
  brokerKeystoreFile=''
  brokerKeystorePass='brokerKeystore::password'
  brokerTruststoreType='JKS'
  brokerTruststoreFile=''
  brokerTruststorePass='brokerTruststore::password'
  httpConnectorPortRange=''
  httpsConnectorPortRange=''
```

Updating the integration node reference to a keystore

About this task

To update the integration node reference to a keystore, use the following command:

```
mqsichangeproperties integrationNodeName -o BrokerRegistry
  -n  brokerKeystoreFile
  -v c:\keystore\server.keystore
```

Where `c:\keystore\server.keystore` is the keystore to be referenced.

Updating the integration node reference to a truststore

About this task

To update the integration node reference to a truststore, use the following command:

```
mqsichangeproperties integrationNodeName -o BrokerRegistry
  -n  brokerTruststoreFile
  -v c:\truststore\server.truststore
```

Where `c:\truststore\server.truststore` is the truststore to be referenced.

Updating the integration node with the keystore password

About this task

Keystores and truststores normally require passwords for access. Use the **mqsisetdbparms** command to add these passwords to the IBM App Connect Enterprise runtime component. To check a password that you have set, use the **mqsireportdbparms** command.

```
mqsireportdbparms integrationNodeName
  -n brokerKeystore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the integration node with the truststore password

About this task

To update the integration node with the truststore password, use the following command:

```
mqsisetdbparms integrationNodeName
  -n brokerTruststore::password
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Updating the integration node with a private key password

About this task

Private keys in the keystore might have their own individual passwords. These can be configured based on the alias name that is specified for the key in the Policy sets and bindings editor. If a key password that is based on the alias is not found, the keystore password is used. The following command updates the integration node with the private key password for the key whose alias is *encKey*.

```
mqsisetdbparms integrationNodeName
  -n brokerTruststore::KeyPassword::encKey
  -u temp -p pa55word
```

The user ID, which can be any value, is not required to access the keystore.

Viewing and setting keystore and truststore runtime properties at integration server level

Configure an integration server to refer to a keystore, a truststore, or both. You must do this configuration before you deploy any message flows that require SSL connections, policy set or bindings for signature, encryption, or X.509 authentication. You must also do this configuration before you enable services that use SSL connections, such as IBM Cloud reporting services.

About this task

An integration server is a named grouping of message flows. You can configure integration servers to be associated with an *integration node* that looks after them, or to run independently of an integration node. For more information about integration servers, see Integration servers and integration nodes.

For integration servers that are managed by an *integration node*, the integration server keystore and truststore runtime property values override equivalent property values on the *integration node*, if any are set.

Keystores can contain two kinds of entries: key entries and trusted certificate entries. If a keystore is used to contain trusted certificates, it is typically referred to as a truststore. IBM App Connect Enterprise can

refer to a keystore and a truststore per integration server. When the integration node or integration server is encrypting or decrypting, it uses entries in its keystore. If the integration node or integration server is verifying a signature or performing X.509 authentication, it uses entries in its truststore.

Displaying integration server level properties

About this task

Integration servers that are managed by integration nodes:

To display the properties of an integration server that is managed by an integration node, run the command:

```
mqsireportproperties integrationNodeName -o ComIbmJVMManager -a -e integration_server
```

Independent integration servers:

To display the properties of an independent integration server, view the <work directory>/server.conf.yaml configuration file.

Updating the integration server reference to a keystore

About this task

Integration servers that are managed by integration nodes:
To update the reference to a keystore for an integration server that is managed by an integration node, use the following command:

```
mqsichangeproperties integrationNodeName -e integration_server -o ComIbmJVMManager
  -n  keystoreFile -v c:\keystore\server.keystore
```

where c:\keystore\server.keystore is the fully qualified path to the Java keystore (JKS), containing the private certificates that is used by the integration server.
Independent integration servers:

To update the reference to a keystore for an independent integration server, edit the <work directory>/server.conf.yaml configuration file.

```
ResourceManagers:
  JVM:
    keystoreFile: 'c:\keystore\server.keystore' # JVM location of the key store
```

where c:\keystore\server.keystore is the fully qualified path to the Java keystore (JKS), containing the private certificates that is used by the integration server.

Updating the integration server reference to a truststore

About this task

Integration servers that are managed by integration nodes:
To update the reference to a truststore for an integration server that is managed by an integration node, use the following command.

```
mqsichangeproperties integrationNodeName -e integration_server -o ComIbmJVMManager
  -n  truststoreFile -v c:\truststore\server.truststore
```

The path, c:\truststore\server.truststore is the fully qualified path to the Java truststore. The Java truststore contains the public certificates that are required by the integration server to establish trust with the services that it connects to.
Independent integration servers:

To update the reference to a truststore for an independent integration server, edit the <work directory>/server.conf.yaml configuration file.

```
ResourceManagers:
  JVM:
    truststoreFile: 'c:\truststore\server.truststore'             # JVM location of the trust
 store
```

where c:\truststore\server.truststore is the fully qualified path to the Java truststore. The Java truststore contains the public certificates that are required by the integration server to establish trust with the services that it connects to.

Updating the keystore and truststore passwords

About this task

Integration servers that are managed by integration nodes:

To update the keystore and truststore passwords for an integration server that is managed by an integration node, use the same commands that are used for setting keystore and truststore runtime properties at integration node level.

- To update the integration node with the keystore password; see "Updating the integration node with the keystore password" on page 2669.
- To update the integration node with the truststore password; see "Updating the integration node with the truststore password" on page 2669.
- To update the integration node with a private key password; see "Updating the integration node with a private key password" on page 2669.

To use the default integration node password for the keystore, the **keystorePass** parameter must be blank, or it must be set to brokerKeystore::password. To use a password other than the default integration node password, use the following commands:

```
mqsichangeproperties integrationNodeName -e integration_server -o ComIbmJVMManager -n
 keystorePass
-v integration_server::keystorePass

mqsisetdbparms integrationNodeName -n integration_server::keystorePass -u na -p password
```

To use the default integration node password for the truststore, the **truststorePass** parameter must be blank, or it must be set to brokerTruststore::password. To use a password other than the default integration node password, use the following commands:

```
mqsichangeproperties integrationNodeName -e integration_server -o ComIbmJVMManager -n
 truststorePass
-v integration_server::truststorePass

mqsisetdbparms integrationNodeName -n integration_server::truststorePass -u na -p password
```

Independent integration servers:

To update the keystore password for an independent integration server, edit the <work directory>/server.conf.yaml configuration file.

```
ResourceManagers:
  JVM:

    #keystorePass: 'keyStorePassword'               # JVM resource alias containing the key
 store password
```

To update the truststore password for an independent integration server, edit the <work directory>/server.conf.yaml configuration file.

```
ResourceManagers:
  JVM:
```

```
    #truststorePass: 'TrustStorePassword'                # JVM resource alias containing the
trust store password
```

Adding new certificates to a keystore or truststore

About this task

If you add new certificates to a keystore or truststore, to ensure that the new certificates are picked up, you must reload the Java virtual machine (JVM). You can reload the JVM by restarting the integration server.

Associating policy sets and bindings with message flows and nodes

Use the BAR file editor to associate policy sets and bindings with message flows and nodes, so that they are available to the integration node at run time.

Before you begin

Use the WS Policy Sets and Bindings editor to create and configure policy sets and bindings (see "Implementing WS-Security" on page 2652).

These instructions also assume that you have packaged your message flow in a BAR file.

About this task

You can associate a policy set with a message flow or with specific message flow nodes. Associations that are made with a flow apply to all message flow nodes described in the Policy Set and Bindings file. Associations at the flow level are defined as being either for consumer or provider nodes.

An association at the message flow node level overrides any association made at the message flow level. You do not enter information about consumer or provider for an association at message flow node level.

Procedure

1. In the IBM App Connect Enterprise Toolkit, open the BAR file that contains the message flow that you want to associate with a policy set and binding.
2. Click the **Manage** tab.
3. Click the message flow or node that you want to associate with a policy set and binding.

   The properties that you can configure for the message flow or for the message flow node are displayed in the **Properties** view.
4. If you are configuring a message flow, enter values in the following fields in the **Properties** view, in the form *{policy_project_name}:policy_set_name* or *{policy_project_name}:policy_set_bindings_name*
   - **Consumer Policy Set**
   - **Consumer Policy Set Bindings**
   - **Provider Policy Set**
   - **Provider Policy Set Bindings**
5. If you are configuring a message flow node, enter values in the following fields in the **Properties** view, in the form *{policy_project_name}:policy_set_name* or *{policy_project_name}:policy_set_bindings_name*
   - **Policy Set**
   - **Policy Set Bindings**

What to do next

For new associations to take effect, deploy the BAR file, then stop and restart the associated message flow.

Exporting a policy set and policy set binding

Use the **mqsireportproperties** command to export a policy set and associated policy set binding to a file.

About this task

This topic shows how to export policy set *myPolicySet* from integration node *myBroker* to a file called *myPolicySet.xml*. The associated binding is *myPolicySetBinding*, which you export to *myPolicySetBinding.xml*.

Procedure

1. Export the policy set to a file:

```
mqsireportproperties myBroker -c PolicySets -o myPolicySet -n ws-security -p myPolicySet.xml
```

2. Export the policy set binding to a file:

```
mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding
   -n ws-security -p myPolicySetBinding.xml
```

What to do next

Make a note of the policy set that is associated to the binding; you will need this information when you import the policy set and binding. This command displays the policy set associated with a binding:

```
mqsireportproperties myBroker -c PolicySetBindings -o myPolicySetBinding -n associatedPolicySet
```

This displays:

```
PolicySetBindings   myPolicySetBinding     associatedPolicySet='myPolicySet'
BIP8071I: Successful command completion.
```

Note: To export a policy set with more than one policy, you must run the **mqsireportproperties** command once for each policy. For example, a policy set might contain a WS-Security policy and a WS-RM policy.

Message flow security and security profiles

IBM App Connect Enterprise provides a security manager for implementing message flow security, so that end-to-end processing of a message through a message flow is secured based on an identity carried in that message instance.

For details of the supported external providers and the operation of the message flow security manager, see “Message flow security overview” on page 2550. For information about the token types that are supported by the SOAP nodes and by external security providers, see “Identity” on page 2554.

When the message flow is a web service implemented by using “SOAP nodes” on page 811 and the identity is to be taken from the “WS-Security” on page 2650 SOAP headers, the SOAP nodes are the Policy Enforcement Point (PEP) and the external provider defined by the “Security profiles” on page 2552 is the Policy Decision Point (PDP).

The following configuration is required to implement message flow security based on an identity carried in `WS_Security` tokens.

- "Policy sets" on page 2662 define the type of tokens used for the identity.
  - To work with a Username and Password identity, configure the policy and binding for Username token "Authentication" on page 2652.
  - To work with a X.509 Certificate identity, configure the policy and binding for X.509 certificate token "Authentication" on page 2652.

    In the Policy Set Binding, set the X.509 certificate Authentication Token `certificates` mode to `Trust Any`. You set it this way (and not to `Trust Store`) so that the certificate is passed to the security provider defined by the Security Profile. Setting it to `Trust Store` will cause the certificate to be validated in the local Integration node Trust Store. For more details, see Policy Sets and Policy Set Bindings editor: Authentication and Protection Tokens panel.
  - To work with a SAML assertion token, configure the policy and binding for SAML token "Authentication" on page 2652.
- The message flow security operation and external provider are defined by the "Security profiles" on page 2552

As an alternative to message flow security and an external PDP, the integration node's truststore can be used as a local PDP for X.509 certificate authentication. For WS-Security signing and encryption using only the local integration node capability, you must configure the integration node's truststore. For details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667, or "Viewing and setting keystore and truststore runtime properties at integration server level " on page 2669.

Kerberos based WS-Security is supported in the SOAP nodes. When you use Kerberos for security, the SOAP node's WS-Security processing links directly with the host's Kerberos infrastructure. The integration node host must be configured for Kerberos, providing a `krb.conf` file to define the Kerberos Key Distribution Center (KDC) and default realm. A Kerberos `keytab` file must also be configured. For more information about configuring Kerberos, see your host's Kerberos documentation.

To work with Kerberos WS-Security in SOAP nodes, create a policy set and bindings specifying Kerberos symmetric encryption tokens on the Message Level Protection panel; see Policy Sets and Policy Set Bindings editor: Message Level Protection panel. Also configure the required settings on the Kerberos settings panel, as described in Policy Sets and Policy Set Bindings editor: Kerberos settings panel, and then associate this policy set and bindings with the SOAP node. You can also associate SOAP nodes with a security profile that sets only propagation, so that Kerberos can be used to:

- Extract the service principal as a Username token from SOAP input nodes
- Propagate the Kerberos Key Distribution Center (KDC) credentials as a Username and password to SOAP request nodes.

WS-Security capabilities

Web service security capabilities are supported by the integration node.

Web service security mechanisms are defined by OASIS standards. See OASIS Standard for WS-Security Specification

For information about the token profile standards, see:

- OASIS Web Services Security Username Token Profile
- OASIS Web Services Security X.509 Certificate Token Profile
- OASIS Web Services Security Kerberos Token Profile

SAML pass-through support is provided, which enables interoperability with WS-Security SAML profiles, without performing subject confirmation processing. This means that it does not provide validation of the trust relationship between the SAML subject and message content signatures. For information about the SAML token profile standards, see:

- OASIS Web Services Security SAML Token Profile 1.1
- SAML V2 Specification

LTPA pass-through support is also provided, which enables LTPA binary tokens to be passed to an external Security Token Service (STS) for processing.

For more information about using the token profiles, see the following topics:

- "Username token capabilities" on page 2675
- "X.509 certificate token capabilities" on page 2677
- "SAML token capabilities" on page 2680
- "Kerberos token capabilities" on page 2682
- "LTPA token capabilities" on page 2684

Username token capabilities

This topic describes WS-Security username token capabilities of the integration node.

For details of using WS-Security username token, see the following capabilities:

- "Username token capabilities for encryption, decryption, signing, and verifying" on page 2675
- "Username token capabilities for authentication and authorization" on page 2675
- "Username token capabilities for identity mapping" on page 2676
- "Username token capabilities for extraction and propagation" on page 2676

Username token capabilities for encryption, decryption, signing, and verifying

For web services, you cannot complete encryption, decryption, signing, and verification by using username tokens.

The username token is not applicable, or supported, for the following in any configuration or direction:

- Encryption
- Decryption
- Signing
- Verification

Username token capabilities for authentication and authorization

For web services, you can complete authentication and authorization by using a Web Services Security username token.

For authentication, the Web Services Security username token must include both the username and the optional password.

The Web Services Security username token "Authentication" on page 2651 and "Authorization" on page 2562 is supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy and binding that defines that a Web Services Security username token is present for authentication; see "Authentication" on page 2652. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 2663.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- LDAP

  Configured by using an LDAP security profile specifying authentication, authorization, or both; see "Creating a security profile for LDAP" on page 2584. For authentication, both a username and password are required.

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588. For authentication, both a username and password are required.

- TFIM V6.1

  Configured by using a TFIM security profile specifying authentication, authorization or both; see "Creating a security profile for TFIM V6.1" on page 2589. For authentication, both a username and password are required.

Username token capabilities for identity mapping

For web services, you can map an identity by using a username token.

"Identity mapping" on page 2564 from a username identity token to a mapped username identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy and binding that defines that a username taken is present. You can use the default policy and binding WSS10Default; see "Default policy set and bindings" on page 2663.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

- TFIM V6.1

  Configured by using a TFIM security profile that specifies identity mapping; see "Creating a security profile for TFIM V6.1" on page 2589.

Username token capabilities for extraction and propagation

This topic describes integration node capability for extraction, propagation, or both using a username token in web services.

The extraction of username into the Properties folder source "Identity" on page 2554 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy and binding which defines that a username taken is present. You can use the default policy and binding WSS10Default; see “Default policy set and bindings” on page 2663.

  Configured with a security profile that defines propagation; see “Creating a security profile” on page 2584

The propagation of a username token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. See “Identity” on page 2554.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  node
  node

  Configured with a security profile that defines propagation; for example, Default Propagation. See “Security profiles” on page 2552

X.509 certificate token capabilities

Various WS-Services Security X.509 certificate token profile standards are supported by IBM App Connect Enterprise.

For details of using the X.509 certificates, see the following capabilities:

- “X.509 certificate token capabilities for encryption” on page 2677
- “X.509 certificate token capabilities for decryption” on page 2678
- “X.509 certificate token capabilities for signing” on page 2678
- “X.509 certificate token capabilities for verifying” on page 2679
- “X.509 certificate token capabilities for authentication” on page 2679
- “X.509 certificate token capabilities for authorization” on page 2679
- “X.509 certificate token capabilities for identity mapping” on page 2680
- “X.509 certificate token capabilities for extraction and propagation” on page 2680

X.509 certificate token capabilities for encryption

For web services, you can complete encryption by using an X.509 certificate token.

X.509 certificate token encryption for providing message “Confidentiality” on page 2652 on outgoing SOAP messages from the integration node is supported in the following configurations:

Capability

- Encrypt (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  node
  node

- Out (provider)

  node

  Configured with a policy set and binding defining the message “Confidentiality” on page 2654.

Trust Store or Policy Decision Point (PDP)

- Integration node Truststore; for more details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667.

Encryption is not supported with external PDPs such as TFIM or LDAP.

X.509 certificate token capabilities for decryption

For web services, you can complete decryption by using an X.509 certificate token.

X.509 certificate token decryption for incoming SOAP message "Confidentiality" on page 2652 is supported in the following configurations:

Capability

- Decrypt (by using an integration node private key)

Policy Enforcement Point (PEP) and direction.

- In (provider)

  node
- In (consumer)

  node
  node

  Configured with a policy set and binding defining the message "Confidentiality" on page 2654.

Trust Store or Policy Decision Point (PDP).

- Integration node Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667.

Decryption is not supported with external PDPs such as TFIM or LDAP.

X.509 certificate token capabilities for signing

For web services, you can use an X.509 certificate token for signing.

X.509 certificate token signing for outgoing SOAP message "Integrity" on page 2652 is supported in the following configurations:

Capability

- Sign (by using an integration node private key)

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  node
  node
- Out (provider)

  node

  Configured with a policy set and binding defining the message "Integrity" on page 2655.

Trust Store or Policy Decision Point (PDP)

- Integration node Truststore; for details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667.

Signing is not supported with an external PDP such as TFIM or LDAP.

X.509 certificate token capabilities for verifying

For web services, you can verify a signing by using an X.509 certificate token profile.

X.509 certificate token verification of the "Integrity" on page 2652 of a signed incoming SOAP message is supported in the following configurations:

Capability

- Verify signature (by using a partner public key)

Policy Enforcement Point (PEP) and direction

- In (provider)

  node
- In (consumer)

  node
  node

  Configured with a policy set and binding defining the message "Integrity" on page 2655

Trust Store or Policy Decision Point (PDP)

- Integration node Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667.

  Signature verification is not supported with an external PDP, such as TFIM or LDAP.

X.509 certificate token capabilities for authentication

For web services, you can complete authentication by using an X.509 certificate token.

The X.509 certificate token "Authentication" on page 2651 of an incoming SOAP message is supported in the following configurations:

Capability

- Authenticate

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a policy set and binding defining the certificate "Authentication" on page 2652.

  Optionally configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- Integration node Trust store; for details, see "Viewing and setting keystore and truststore runtime properties at integration node level" on page 2667.
- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.
- TFIM V6.1

  Configured by using a TFIM security profile specifying authentication; for details, see "Creating a security profile for TFIM V6.1" on page 2589.

Certificate authentication with an external LDAP PDP is not supported.

X.509 certificate token capabilities for authorization

The X.509 certificate token is not supported for authorization in any configuration or direction.

X.509 certificate token capabilities for identity mapping

For web services, you can map an identity by using an X.509 certificate token.

The integration node supports "Identity mapping" on page 2564 from an X.509 certificate token in an incoming SOAP message header to username tokens in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a policy set and binding defining the certificate "Authentication" on page 2652.

  Configured with a security profile defining an external Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- TFIM V6.1

  Configured by using a TFIM security profile specifying identity mapping; for details, see "Creating a security profile for TFIM V6.1" on page 2589.

- WS-Trust v1.3 STS (TFIM V6.2)

  Configured by using a WS-Trust v1.3 STS security profile specifying identity mapping; for details, see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

Identity mapping is not supported with LDAP, or at outbound nodes.

Username tokens only can be propagated.

X.509 certificate token capabilities for extraction and propagation

This topic describes integration node web services capability for extraction and propagation X.509 certificate token.

The integration node does not support propagation of an X.509 certificate.

The X.509 certificate token extraction is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a policy set and binding defining the X.509 certificate is present; see "Implementing WS-Security" on page 2652.

  Configured with a security profile defining propagation; see the "Security profiles" on page 2552.

SAML token capabilities

This topic describes WS-Security SAML token capabilities of the integration node.

The integration node provides SAML pass-through support, which means that the mechanisms for subject confirmation are not enforced. The SAML token is extracted and passed to an external Security Token Service (STS) for validation. The STS to be used is specified on a security profile.

For information about using WS-Security SAML tokens, see the following topics:

- "SAML token capabilities for encryption, decryption, signing, and verifying" on page 2681

- "SAML token capabilities for authentication and authorization" on page 2681
- "SAML token capabilities for identity mapping" on page 2681
- "SAML token capabilities for extraction and propagation" on page 2682

SAML token capabilities for encryption, decryption, signing, and verifying

The integration node provides SAML pass-through support, which means that the encryption, decryption, signing, and verifying mechanisms for achieving SAML subject confirmation are not enforced by the broker.

SAML token capabilities for authentication and authorization

For web services, you can complete authentication and authorization using a SAML token.

The SAML token "Authentication" on page 2651 and "Authorization" on page 2562 are supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and binding that defines that a SAML pass-through 1.1 or SAML pass-through 2.0 token is present for authentication; see "Authentication" on page 2652. The integration node provides only SAML pass-through support, which means that the SAML token is extracted and passed to an external Security Token Service (STS) for validation. The STS to be used is specified in a security profile. The STS processing can be used to implement authentication based on the SAML principal, and authorization based on SAML attributes.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see "Creating a security profile for WS-Trust V1.3 (TFIM V6.2)" on page 2588.

SAML token capabilities for identity mapping

This topic describes the integration node web services capability for identity mapping using a SAML token.

"Identity mapping" on page 2564 from a SAML identity token to a mapped SAML identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and bindings that specifies a SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see “Creating a security profile for WS-Trust V1.3 (TFIM V6.2)” on page 2588.

SAML token capabilities for extraction and propagation

This topic describes integration node capability for extraction, propagation, or both using a SAML token in web services.

The extraction of a SAML token into the Properties folder source “Identity” on page 2554 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and bindings that specifies a SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

  Configured with a security profile that defines propagation; see “Creating a security profile” on page 2584

The propagation of a SAML token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. For more information, see “Identity” on page 2554.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  node
  node

  Configured with a security profile that defines propagation. For more information, see “Security profiles” on page 2552.

  Configured with a security policy set and bindings that specifies SAML pass-though 1.1 or SAML pass-through 2.0 authentication token.

Kerberos token capabilities

This topic describes WS-Security Kerberos token capabilities of the integration node.

For details of using WS-Security Kerberos tokens, see the following topics:

- “Kerberos token capabilities for encryption, decryption, signing, and verifying” on page 2682
- “Kerberos token capabilities for authentication and authorization” on page 2683
- “Kerberos token capabilities for identity mapping” on page 2683
- “Kerberos token capabilities for extraction and propagation” on page 2684

Kerberos token capabilities for encryption, decryption, signing, and verifying

You can use Kerberos tokens for encryption, decryption, signing, and verifying.

Kerberos token encryption for providing message “Confidentiality” on page 2652 and “Integrity” on page 2652 on outgoing SOAP messages from the integration node is supported in the following configurations:

Capability

- Encrypt, by using a Kerberos Key Distribution Center (KDC)
- Decrypt, by using the Kerberos `keytab` file

Policy Enforcement Point (PEP) and direction

- In (provider)

  node
- In (consumer)

  node
  node

  Configured with a policy set and binding defining the message "Integrity" on page 2655
- Out (consumer)

  node
  node
- Out (provider)

  node

  Configured with a Kerberos policy set and binding.

Trust Store or Policy Decision Point (PDP)

- Kerberos KDC.

Kerberos token capabilities for authentication and authorization

This topic describes the integration node web services capability for authentication, authorization, or both using a Kerberos token.

Kerberos is not applicable to authorization. Kerberos token encryption for providing message "Authentication" on page 2652 on outgoing SOAP messages from the integration node is supported in the following configurations:

Capability

- Authenticate using a Kerberos `keytab` file.

Policy Enforcement Point (PEP) and direction

- In (provider)

  node
- In (consumer)

  node
  node

  Configured with a policy set and binding defining the message "Integrity" on page 2655

Trust Store or Policy Decision Point (PDP)

- Kerberos `keytab` file.

Kerberos token capabilities for identity mapping

This topic describes integration node web services capability for identity mapping using a Kerberos token.

Kerberos tickets from SOAP nodes are not supported for token mapping with an external Security Token Service (STS) configured in the security profile.

On the Inbound route, with SOAPInput and SOAPAsyncResponse nodes, the presence of a security profile with propagation enabled causes the Kerberos Service Principal Name (SPN) to be placed in the properties tree as a Username token.

On the Outbound route, with SOAPRequest and SOAPAsyncRequest nodes, identity propagation can be used to provide the Kerberos Key Distribution Center (KDC) credentials. Arrange for the KDC credentials to be set as a Username and password token in the properties tree and associate the SOAP node with a security profile that specifies propagation; otherwise the KDC credentials are obtained using the Kerberos resource credentials that are created using the **mqsisetdbparms** command.

Kerberos token capabilities for extraction and propagation

This topic describes integration node capability for extraction, propagation, or both using a Kerberos token in web services.

Kerberos tickets from SOAP nodes are not supported for token extraction and propagation with an external Security Token Service (STS) configured in the security profile.

On the Inbound route, with SOAPInput and SOAPAsyncResponse nodes, the presence of a security profile with propagation enabled causes the Kerberos Service Principal Name (SPN) to be placed in the properties tree as a Username token.

On the Outbound route, with SOAPRequest and SOAPAsyncRequest nodes, identity propagation can be used to provide the Kerberos Key Distribution Center (KDC) credentials. Arrange for the KDC credentials to be set as a Username and password token in the properties tree and associate the SOAP node with a security profile that specifies propagation; otherwise the KDC credentials are obtained using the Kerberos resource credentials that are created using the **mqsisetdbparms** command.

LTPA token capabilities

This topic describes WS-Security LTPA token capabilities of the integration node.

The integration node provides LTPA pass-through support, which means that the LTPA token is extracted and passed to an external Security Token Service (STS) for validation. The STS to be used is specified on a security profile.

For information about using WS-Security LTPA tokens, see the following topics:

- "LTPA token capabilities for encryption, decryption, signing, and verifying" on page 2684
- "LTPA token capabilities for authentication and authorization" on page 2684
- "LTPA token capabilities for identity mapping" on page 2685
- "LTPA token capabilities for extraction and propagation" on page 2685

LTPA token capabilities for encryption, decryption, signing, and verifying

This topic describes integration node web services capability for encryption, decryption, signing, and verifying using LTPA tokens.

The LTPA token is not applicable, or supported, for the following in any configuration or direction:

- Encryption
- Decryption
- Signing
- Verifying

LTPA token capabilities for authentication and authorization

For web services, you can complete authentication and authorization using an LTPA token.

The LTPA token "Authentication" on page 2651 and "Authorization" on page 2562 are supported only in the following configuration:

Capability

- Authenticate
- Authorize

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and binding that defines that an LTPA token is present for authentication; see “Authentication” on page 2652. The integration node provides only LTPA pass-through support, which means that the LTPA token is extracted and passed to an external Security Token Service (STS) for validation. The STS to be used is specified in a security profile. The STS processing can be used to implement authentication and authorization based on the LTPA principal and realm.

  Configured with a security profile defining the Policy Decision Point (PDP); see the PDP section that follows.

Trust Store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile specifying authentication, authorization or both; see “Creating a security profile for WS-Trust V1.3 (TFIM V6.2)” on page 2588.

LTPA token capabilities for identity mapping

This topic describes the integration node web services capability for identity mapping using an LTPA token.

“Identity mapping” on page 2564 from or to an LTPA identity token is supported only in the following configurations:

Capability

- Identity mapping

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and bindings that specifies an LTPA pass-through authentication token.

  Configured with a security profile defining the external Policy Decision Point (PDP); see the PDP section that follows.

Trust store or PDP

- WS-Trust v1.3 STS

  Configured by using a WS-Trust v1.3 STS security profile that specifies identity mapping; see “Creating a security profile for WS-Trust V1.3 (TFIM V6.2)” on page 2588.

LTPA token capabilities for extraction and propagation

This topic describes integration node capability for extraction, propagation, or both using an LTPA token in web services.

The extraction of an LTPA token into the Properties folder source “Identity” on page 2554 fields, is supported in the following configurations:

Capability

- Extraction

Policy Enforcement Point (PEP) and direction

- In (provider)

  node

  Configured with a security policy set and bindings that specifies an LTPA pass-though authentication token.

  Configured with a security profile that defines propagation; see “Creating a security profile” on page 2584

The propagation of an LTPA token into the SOAP WS-Security header, from the token present in either the mapped or the source identity fields in the properties folder, is supported in the following configuration. For more information, see “Identity” on page 2554.

Capability

- Propagate

Policy Enforcement Point (PEP) and direction

- Out (consumer)

  node
  node

  Configured with a security profile that defines propagation. For more information, see “Security profiles” on page 2552.

  Configured with a security policy set and bindings that specifies an LTPA pass-through authentication token.

Chapter 9. Performance, monitoring, and workload management

You can change various aspects of your configuration to tune integration servers and message flows, and to monitor message flows and publish/subscribe applications. You can also send logging information to an Elasticsearch, Logstash, and Kibana (ELK) server, and view the data in a Kibana dashboard.

About this task

Performance

How you configure your IBM App Connect Enterprise environment can affect the performance of your applications. For more information about these options, see “Performance” on page 2688.

Monitoring

You can configure your integration nodes or message flows to generate data and statistics that you can use to assess behavior and performance. For more information about these options, see “Message flow monitoring” on page 2771.

Statistics and accounting

You can collect message flow statistics and accounting data to record performance and operating details of one or more message flows. For more information, see “Message flow statistics and accounting data” on page 2699.

Recording and viewing messages

You can configure a message flow to capture some of the data that it processes. You can then view this data from the web user interface or programmatically. For more information, see “Recording, viewing, and replaying data” on page 2834.

Activity logging

Activity logs help you understand what your message flows are doing by providing a high-level overview of how IBM App Connect Enterprise interacts with external resources. For more information about activity logs, see “Activity Logs” on page 2854.

Reporting logs and events to a Logstash input in an ELK stack

You can configure integration servers to send the following information to a Logstash input in an Elasticsearch, Logstash, and Kibana (ELK) stack, so that you can view the data in a Kibana dashboard:

- Logging information (BIP messages)
- Message flow monitoring events
- Activity logging events

For more information, see “Reporting logs and monitoring events to a Logstash input in an ELK stack” on page 2869 and “Configuring integration servers to send logs and events to Logstash in an ELK stack” on page 2870.

Workload management

System administrators can monitor and control the speed that messages are processed within message flows. For more information, see “Workload management” on page 2873.

Performance

You can use message flow design and coding techniques to optimize the performance of your message flows. You can then analyze statistical information for your message flows and modify aspects of your integration server configuration to tune the performance of your integration servers and message flows.

About this task

The way in which you configure your integration server environment can affect the performance of your applications. Review the information in "Performance planning" on page 16, and decide which actions you can take to improve performance.

The following topics describe some design tips and techniques for optimizing performance, and explain how you can collect message flow and resource statistics and then use the data to help you tune performance:

- "Message flow design and performance" on page 2689
- "Code design and performance" on page 2690
- "Analyzing message flow performance" on page 2699
- "Analyzing resource performance" on page 2739

Activity Logs also provide information that you might find helpful when analyzing and tuning the performance of your integration servers. For more information, see "Activity Logs structure and types" on page 2858.

The following IBM Support pages provide results of a competitive performance evaluation of IBM App Connect Enterprise Version 11.0.0 against IBM Integration Bus Version 10.0.0:

- IBM App Connect Enterprise V11 performance reports

  Results of a competitive performance evaluation of IBM App Connect Enterprise Version 11.0.0.6 against IBM Integration Bus Version 10.0.0.18 in traditional non-virtualized environments on Linux, Windows, and AIX.
- IBM App Connect Enterprise V11 container performance reports

  Results of a competitive performance evaluation of IBM App Connect Enterprise Version 11.0.0.4 against IBM Integration Bus Version 10.0.0.16 in Docker container environments on Linux and Windows.

Performance planning

When you design your IBM App Connect Enterprise environment and the associated resources, the decisions that you make can affect the performance of your applications.

About this task

If you are planning to create and maintain a small, stand-alone system with limited requirements for availability and performance, it is possible to achieve this relatively quickly. However, if you are planning a large or complex system, or if you have specific requirements for high availability or performance, it is worth taking time to understand the factors that can influence your design and the techniques that you can use to optimize it. As a starting point, consider the following aspects of your system and understand how these factors can influence performance:

Message flows
: A message flow includes an input node that receives a message from an application over a particular protocol; for example, IBM MQ. The message must be parsed by the input node, and the performance impact of this parsing varies according to the parser that is used and the number of parses required. You can reduce the impact of parsing by using some optimization techniques such as parsing avoidance or partial parsing. For more information about parsing, see "Parsing and message flow performance" on page 2697.

The number and length of message flows have implications for performance, and it is important to keep the number of nodes to a minimum. See “Message flow design and performance” on page 2689 for more information about optimizing message flow performance. Other aspects of processing in a message flow that might affect performance are the amount, efficiency, and complexity of ESQL, access to databases, and how many message tree copies are made. For more guidance about these factors, see “Code design and performance” on page 2690.

It is also important to consider how you split your business logic; how much work should the application do, and how much should the message flow do? Every interaction between an application and a message flow involves I/O and message parsing, and therefore adds to processing time. Design your message flows, and design or restructure you applications, to minimize these interactions.

Messages and message models

The type, format, and size of the messages that are processed can have a significant effect on the performance of a message flow. For example, if you process persistent messages, they must be stored for safekeeping.

If you do not plan to interrogate the structure, you can use the BLOB domain.

If you are working with general text or binary messages, then you need to create a DFDL (or MRM) model to describe the message. The organization of the model can have a significant effect on performance. For fields that are choices, are optional, or are in arrays, the model must identify the field as efficiently as possible. Use tags (named initiators in DFDL) if they are available. To resolve a choice, DFDL provides a direct dispatch mechanism, which avoids parsing each branch in turn if there is a field that indicates which branch to take. If you are using a DFDL discriminator, ensure that the discriminator is placed so that it is evaluated as early as possible. While regular expressions can be used to identify and extract fields, they are generally slower than other techniques.

If you are working in XML, be aware that it can be verbose, and therefore produce large messages. However, XML message content is easier to understand than other formats, such as binary fixed-length format.

For more information about these factors, see Performance considerations for regular expressions in TDS messages.

Integration node configuration

You can create and configure one or more integration nodes, on one or more computers, and for each integration node you can create multiple integration servers, and multiple message flows. Your configuration decisions can influence message flow performance, and how efficiently messages can be processed.

For more information about these factors, see “Tuning the integration node” on page 2762, and “Optimizing message flow throughput” on page 2763.

Message flow design and performance

Consider the performance implications arising from the design of your message flow.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

To achieve the best performance from your message flows, consider the following points:

- Ensure that message flows are an appropriate length. Long message flows take longer to execute than short ones, but the use of many short message flows reduces performance more than the use of a few longer message flows.

- Use the minimum number of nodes required, and ensure that any subflows or loops are used appropriately.
- Avoid having consecutive ESQL Compute nodes in a message flow with no other nodes between them. Combine the logic into a single Compute node instead. Also avoid having consecutive JavaCompute nodes in the message flow.
- Minimize the volume of message parsing and adopt the most efficient parsing strategy; for more information, see “Parsing and message flow performance” on page 2697.
- Custom canonical data formats provide a point of standardization, but they also involve additional CPU processing. The use of such formats typically results in two additional conversions per invocation of a message flow: from the external format into the canonical format and then from the canonical format to the final external format.
- If a message flow makes a synchronous call to an external application or service that can be slow or unpredictable in its response, it is more efficient to write the message flow using an asynchronous model.
- Ensure that your processing logic implements the coding tips provided in “Code design and performance” on page 2690.

Code design and performance

Consider the performance implications arising from the design of your code.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

The following topics describe some of the design decisions that can influence the performance of your code:

- “ESQL code tips” on page 2690
- “Java code tips” on page 2694
- “XPath and XSLT code tips” on page 2696
- “Parsing and message flow performance” on page 2697
- “Message tree navigation and copying” on page 2698

ESQL code tips

You can improve message flow performance with ESQL by using some optimization techniques.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

When you write your ESQL code, you can use several techniques to improve the performance of your message flows. The following sections contain guidance about how to improve the performance of your ESQL code:

- “ESQL array processing” on page 2691

ESQL array processing

About this task

Array subscripts [ ] are expensive in terms of performance because of the way in which subscript is evaluated dynamically at run time. By avoiding the use of array subscripts wherever possible, you can improve the performance of your ESQL code. You can use reference variables instead, which maintain a pointer into the array and which can then be reused; for example:

```
DECLARE myref REFERENCE TO InputRoot.XML.Invoice.Purchases.Item[1];
-- Continue processing for each item in the array
WHILE LASTMOVE(myref)=TRUE DO
   -- Add 1 to each item in the array
   SET myref = myref + 1;
   -- Do some processing
   -- Move the dynamic reference to the next item in the array
   MOVE myref NEXTSIBLING;
END WHILE;
```

ESQL array processing example:

The following example shows ESQL being used to process records read from a database. The repeated use of array subscripts such as `Environment.Variables.DBData[A]` increases the processing time significantly:

```
SET Environment.Variables.DBDATA[] =
(
SELECT T.*
FROM Database.{'ABC'}.{'XYZ'} as T
);

DECLARE A INTEGER 1;
DECLARE B INTEGER CARDINALITY(Environment.Variables.*[]);
SET JPcntFODS = B;
WHILE A <= B DO
            CALL CopyMessageHeaders();
            CREATE FIELD OutputRoot.XML.FODS;
            DECLARE outRootRef REFERENCE TO OutputRoot.XML.Data;

            SET outRootRef.Field1 = Trim(Environment.Variables.DBDATA[A].Field1);
            SET outRootRef.Field2 = Trim(Environment.Variables.DBDATA[A].Field2);
            SET outRootRef.Field3 = Trim(Environment.Variables.DBDATA[A].Field3);
            SET outRootRef.Field4 = Trim(Environment.Variables.DBDATA[A].Field4);
            SET outRootRef.Field5 = Trim(Environment.Variables.DBDATA[A].Field5);
            . . .
            . . .
            SET outRootRef.Field37 = CAST(Environment.Variables.DBDATA[A].Field37)

            SET A = A + 1;
            PROPAGATE;
END WHILE;
```

You can reduce the processing time significantly by using reference variables

ESQL CARDINALITY function

About this task

Avoid the use of CARDINALITY in a loop; for example:

```
WHILE ( I < CARDINALITY (InputRoot.MRM.A.B.C[]
```

The CARDINALITY function must be evaluated each time the loop is traversed, which is costly in performance terms. This is particularly true with large arrays because the loop is repeated more frequently. It is more efficient to determine the size of the array before the WHILE loop (unless it changes in the loop) so that it is evaluated only once; for example:

```
SET ARRAY_SIZE = CARDINALITY (InputRoot.MRM.A.B.C[]
WHILE ( I < ARRAY_SIZE )
```

ESQL DECLARE and EVAL statements

About this task

Reduce the number of DECLARE statements (and therefore the performance cost) by declaring a variable and setting its initial value within a single statement. Alternatively, you can declare multiple variables of the same data type within a single ESQL statement rather than in multiple statements. This technique also helps to reduce memory usage.

The EVAL statement is sometimes used when there is a requirement to dynamically determine correlation names. However, it is expensive in terms of CPU use, because it involves the statement being run twice. The first time it runs, the component parts are determined, in order to construct the statement that will be run; then the statement that has been constructed is run.

ESQL PASSTHRU statement

About this task

The following techniques can significantly improve performance when you are using PASSTHRU statements:

- Avoid the use of the PASSTHRU statement with a CALL statement to invoke a stored procedure. As an alternative, you can use the `CREATE PROCEDURE ... EXTERNAL ...` and `CALL ...` commands.
- When you are working with SQL statements that require literal or data values, use host variables, which map a column value to a variable. This enables dynamic SQL statements to be reused within the database. An SQL PREPARE on a dynamic statement is an expensive operation in performance terms, so it is more efficient to run this only once and then EXECUTE the statement repeatedly, rather than to PREPARE and EXECUTE every time.

For example, the following statement has two data and literal values, `100` and `IBM`:

```
PASSTHRU('UPDATE SHAREPRICES AS SP SET Price = 100 WHERE SP.COMPANY = ‘IBM'');
```

This statement is effective when the price is `100` and the company is `IBM`. When either the *Price* or *Company* changes, another statement is required, with another SQL PREPARE statement, which impacts performance.

However, by using the following statement, *Price* and *Company* can change without requiring another statement or another PREPARE:

```
PASSTHRU('UPDATE SHAREPRICES AS SP SET Price = ? WHERE SP.COMPANY = ?',
InputRoot.XML.Message.Price,InputRoot.XML.Message.Company);
```

You can check that dynamic SQL is achieving maximum statement reuse, by using the following commands to display the contents of the SQL statement cache in DB2:

```
db2 connect to <database name>
db2 get snapshot for database on <database name>
```

Use the following commands to see the contents of the dynamic statement cache:

```
db2 connect to <database name>
db2 get snapshot for dynamic SQL on <database name>
```

ESQL reference variables

About this task

You can use reference variables to refer to long correlation names such as `InputRoot.XMLNSC.A.B.C.D.E`. Declare a reference pointer as shown in the following example:

```
DECLARE refPtr REFERENCE to InputRoot.XMLNSC.A.B.C.D;
```

To access element E of the message tree, use the correlation name `refPtr.E`.

You can use REFERENCE and MOVE statements to help reduce the amount of navigation within the message tree, which improves performance. This technique can be useful when you are constructing a large number of SET or CREATE statements; rather than navigating to the same branch in the tree, you can use a REFERENCE variable to establish a pointer to the branch and then use the MOVE statement to process one field at a time.

ESQL string functions

About this task

String manipulation functions used within ESQL can be CPU intensive; functions such as LENGTH, SUBSTRING, and RTRIM must access individual bytes in the message tree. These functions are expensive in performance terms, so minimizing their use can help to improve performance. Where possible, also avoid executing the same concatenations repeatedly, by storing intermediate results in variables.

Message trees with repeating records

About this task

Performance can be reduced under the following conditions:

- You are using ESQL processing to manipulate a large message tree
- The message tree consists of repeating records or many fields
- You have used explicit SET statements with field reference paths to access or create the fields
- You have observed a gradual slowing of message flow processing as the ESQL processes more fields or repetitions

This problem occurs when you use field references, rather than reference variables, to access or create consecutive fields or records.

The following example shows independent SET statements using field reference paths to manipulate the message tree. The SET statement takes a source and target parameter, where either or both parameters are field references:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field = '1';
```

Performance is affected by the SET statement being used to create many more fields, as shown in the following example:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field2 = '2';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field3 = '3';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field4 = '4';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field5 = '5';
```

In this example, the five fields that are created are all children of ParentA. Before the specified field can be created or modified, the integration node must navigate the named message tree to locate the point in the message tree that is to be altered. For example:

- To access field 1, the SET statement navigates to ParentA, then to the first field, involving two navigations.
- To access field 5, the SET statement navigates to ParentA, then traverses each of the previous fields until it reaches field 5, involving six navigations.

Navigating over all the fields that precede the specified field causes the loss in performance.

The following example shows repeating fields being accessed in an input message tree:

```
DECLARE myChar CHAR;
DECLARE thisRecord INT 0;
WHILE thisRecord < 10000 DO
 SET thisRecord = thisRecord + 1;
 SET myChar = InputRoot.MRM.myParent.myRepeatingRecord[thisRecord];
END WHILE;
```

When index notation is used, as the count increases, the processing must navigate over all the preceding fields to get the required field; it must count over the previous records to get to the one that is represented by the current indexed reference.

- When accessing InputRoot.MRM.myParent.myRepeatingRecord[1], one navigation takes place to get to the first record
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[2], two navigations take place to get to the second record
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[N], *N* navigations take place to get to the N-th record

Therefore, the total number of navigations for this WHILE loop is: 1 + 2 + 3 + ... + *N*, which is not linear.

If you are accessing or creating consecutive fields or records, you can solve this problem by using reference variables.

When you use reference variables, the statement navigates to the main parent, which maintains a pointer to the field in the message tree. The following example shows the ESQL that can be used to reduce the number of navigations when creating new output message tree fields:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
DECLARE outRef REFERENCE TO OutputRoot.XMLNS.TestCase.StructureA.ParentA;
SET outRef.field2 = '2';
SET outRef.field3 = '3';
SET outRef.field4 = '4';
SET outRef.field5 = '5';
```

When referencing repeating input message tree fields, you can use the following ESQL:

```
DECLARE myChar CHAR;
DECLARE inputRef REFERENCE TO InputRoot.MRM.myParent.myRepeatingRecord[1];
WHILE LASTMOVE(inputRef) DO
 SET myChar = inputRef;
 MOVE inputRef NEXTSIBLING NAME 'myRepeatingRecord';
END WHILE;
```

For further information, see “Creating dynamic field references” on page 1639.

Java code tips

You can improve message flow performance with Java by using some optimization techniques.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

A significant proportion of the performance problems relating to Java are caused by memory and garbage collection issues, by interaction with external resources, and by Java code that could benefit from optimization. Follow these steps to identify and solve such performance issues:

- Use the JVM resource statistics to review how much memory is in use by the JVM, and how often garbage collection is occurring in the integration server. For more information, see “Resource statistics data: Java Virtual Machine (JVM)” on page 2754.
- If you are calling external resources, review the elapsed time of the call.
- Review your Java code to identify any aspects that might benefit from optimization. There are several techniques that you can use to optimize the performance of your Java code in a message flow, including those described in the following sections:
  - “Tree references” on page 2695
  - “Java string concatenation” on page 2696
  - “Java BLOB processing” on page 2696
  - “Java manual garbage collection” on page 2696

The JavaCompute node enables you to include any valid Java processing, and it is common to reuse existing Java classes for specific functions. However, this code impacts the performance of the message flow, so ensure that any code used in this way is efficient.

Where possible, use the supplied nodes and functions of IBM App Connect Enterprise to perform processing on a message, rather than re-creating function that is already provided. For example, do not use a custom XML parser within the JavaCompute node. The XMLNSC parser that is provided with IBM App Connect Enterprise is a high-performing parser that also offers validation against a schema, and the message tree that it generates can be used throughout the message flow. This is not the case with a custom parser with its own data structures.

Avoid starting new processes in the JavaCompute node to perform a specific piece of work. This is expensive in performance terms and results in additional CPU costs and an increased load on the system, because many short-lived processes are started and terminated when the work is completed. One of the benefits of the IBM App Connect Enterprise runtime is that processes are long lived and initialization is minimized. This reduces total processing costs, because new processes are not started each time the message flow is invoked.

For information about tools that you can use to assess the current status of a running Java application, see developerWorks IBM Monitoring and Diagnostic Tools for Java - Health Center Version 2.1.

Tree references

About this task

Avoid building and navigating trees without storing intermediate references. For example, the following code repeatedly navigates from `root` to build the tree:

```
MbMessage newEnv = new MbMessage(inAssembly.getMessage());
newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME, "Destination", null);
newEnv.getRootElement().getFirstChild().createElementAsFirstChild(MbElement.TYPE_NAME,
 "MQDestinationList", null);
newEnv.getRootElement().getFirstChild().getFirstChild().createElementAsFirstChild(MbElement.TYPE_NAME,"Destin
 null);
```

However, it is more efficient to store references as shown in the following example:

```
MbMessage newEnv = new MbMessage(inAssembly.getMessage());
```

```
MbElement destination =
 newEnv.getRootElement().createElementAsFirstChild(MbElement.TYPE_NAME,"Destination", null);
MbElement mqDestinationList = destination.createElementAsFirstChild(MbElement.TYPE_NAME,
 "MQDestinationList",
    null);
mqDestinationList.createElementAsFirstChild(MbElement.TYPE_NAME,"DestinationData", null);
```

Java string concatenation

About this task

The + operator has a negative impact on performance because it involves creating a new String object for each concatenation; for example:

```
keyforCache = hostSystem + CommonFunctions.separator
+ sourceQueueValue + CommonFunctions.separator
+ smiKey + CommonFunctions.separator
+ newElement;
```

To improve performance, use the StringBuffer class and append method to concatenate java.lang.String objects, rather than the + operator, as shown in the following example:

```
StringBuffer keyforCacheBuf = new StringBuffer();
keyforCacheBuf.append(hostSystem);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(sourceQueueValue);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(smiKey);
keyforCacheBuf.append(CommonFunctions.separator);
keyforCacheBuf.append(newElement);
keyforCache = keyforCacheBuf.toString();
```

Java BLOB processing

About this task

If you want to process a BLOB (by cutting it into chunks or by inserting characters, for example), it is efficient to use a JavaCompute node with its strong processing capabilities. If you are using a JavaCompute node, use ByteArrays and ByteArrayOutputStream to process the BLOB.

Java manual garbage collection

About this task

Avoid manually invoking garbage collection, because it suspends the processing in the JVM, which significantly inhibits message throughput. In one example, a Java-based transformation message flow that was processing at the rate of 1300 messages per second without manual garbage collection was reduced to 100 messages per second when one manual garbage collection call was added to each message being processed.

XPath and XSLT code tips

You can improve message flow performance with XPath and XSLT by using some optimization techniques.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

You can improve the performance of XPath by explicitly specifying the number of instances that are required. For example:

- `/element[1]` finds the first occurrence of the element and then stops
- `/element[2]` finds the second occurrence of the element and then stops
- `/element` returns a node set of all instances in the message, which involves a full parse of the message

For more information about techniques that you can use to optimize your XML code, see Top 10 tips for using XPath.

XSLT has advantages in terms of its potential for code reuse, and caching is available for loaded stylesheets in IBM App Connect Enterprise. However, the stylesheet requires a BLOB as input and produces a BLOB as output, and it is less efficient than ESQL and Java in terms of interaction with the message tree. When it is used mid-stream with other IBM App Connect Enterprise technologies, it results in increased message parsing and serialization.

You can optimize the performance of your XSLT code by using a templates object (with different transformers for each transformation) to do multiple transformations with the same set of stylesheet instructions. You can also improve the efficiency of your stylesheets, by using the following techniques:

- Use `xsl:key` elements and the `key()` function as an efficient way to retrieve node sets
- Where possible, use pattern matching rather than `xsl:if` or `xsl:when` statements
- Avoid the use of `"//"` (descendant axes) patterns near the root of a large document

Also consider the following points:

- When you are creating variables, `<xsl:variable name="abcElem" select="abc"/>` is usually faster than `<xsl:variable name="abcElem"><xsl:value-of-select="abc"/></xsl:variable>`
- `xsl:for-each` is fast because it does not require pattern matching
- `xsl:sort` prevents incremental processing
- Use of index predicates within match patterns can be costly in terms of performance
- Decoding and encoding are costly in terms of performance
- If the XSLT is cached, the performance is improved because the XSLT is not parsed every time it is used; for more information, see node

Parsing and message flow performance

Understand the impact of parsing on message flow performance, and use techniques to limit the effect and improve performance.

Before you begin

Read the following topics:

- “Performance” on page 2688
- “Performance planning” on page 16

About this task

Parsing is the means of populating and serializing the message tree from the data, and it can occur whenever the message body is accessed. Multiple parsers are available, and the message complexity varies together with the cost in terms of performance. You can minimize the impact of parsing on message flow performance in the following ways:

Procedure

- Identify the message type as quickly as possible, and avoid using multiple parses to find it.
- Use the most efficient parser available, such as XMLNSC for XML parsing, and DFDL for non-XML parsing. For more information about parsing, see "Parsers" on page 520 and Available parsers.
- Use parser optimization techniques, such as parsing avoidance, partial parsing (parsing on demand), and opaque parsing:
  - If possible, avoid the use of parsing completely, or consider sending only changed data. If you promote or copy key data structures to MQMD, MQRFH2, or JMS properties, you might be able to avoid the need to parse the user data; this technique can be particularly useful for message routing.
  - Partial parsing (known as parsing on demand) involves parsing a message only when it is necessary to resolve the reference to a particular part of its content. For more information, see Parsing on demand.
  - Opaque parsing reduces the size of the message tree, which results in improved performance. You can configure opaque parsing on the **Parser Options** tab of input nodes. If you enable validation, you cannot use opaque parsing.

Message tree navigation and copying

Message tree navigation and message tree copying can reduce message flow performance, so it is important to use them appropriately and limit their usage where possible.

Before you begin

Read the following topics:

- "Performance" on page 2688
- "Performance planning" on page 16

About this task

Long paths are inefficient, so ensure that you minimize their usage, particularly in loops. Using reference variables and pointers in ESQL and Java can improve performance. Where possible, build a smaller message tree by using compact parsers such as DFDL, XMLNSC, MRM XML, RFH2C, and use opaque parsing.

Message tree copying is also costly in terms of resources, because it causes the logical tree to be duplicated in memory. You can improve performance by reducing the number of times that the message tree is copied, by using the following techniques:

- Reduce the number of Compute and JavaCompute nodes in a message flow
- If possible, set the `Compute Mode` property on the node to not include the message
- Copy at an appropriate level in the message tree; for example, copy once rather than for multiple branch nodes
- Copy data to the environment; however, remember that changes are not backed out

In addition to these techniques, the effect of copying the message tree is reduced if you also reduce the size of the message tree by using compact parsers and opaque parsing. For more information, see "Parsing and message flow performance" on page 2697.

Analyzing message flow performance

You can configure your integration servers to collect statistics about the operation and behavior of your message flows. Use this information to assess the performance of your message flows.

Before you begin

Read the concept topic "Message flow statistics and accounting data" on page 2699

About this task

Measures of performance typically include product speed in terms of processing rate and response times, and resource usage in terms of the CPU and memory consumed. In order to assess the performance of an application, metrics are used to compare the actual performance with the required or expected performance, using measures such as the number of messages per second, elapsed time, CPU utilization, or CPU cost per message. You can use the message flow statistics and accounting data to assess the performance of your message flows and applications using some of these metrics, and to identify the message flows and nodes that might benefit from tuning to improve performance.

Message flow statistics for subflows are accumulated in the metrics for the top-level parent flow. For example, if a subflow contains an Input node, the number of message flow threads and any node-level additional instances from the subflow will be reported in the **NumberOfThreadsInPool** property of the statistics for the parent flow.

You can configure and start or stop the collection of statistics data, either by modifying the `.yaml` configuration file for the integration node or server, or by running the command. For more information, see "Configuring the collection of message flow statistics by using a .yaml configuration file" on page 2729 and "Configuring the collection of message flow statistics by using the mqsichangeflowstats command" on page 2732.

You can view the collected statistics by using the web user interface, as described in "Viewing message flow statistics and accounting data" on page 2738.

For information about how to see the current settings for message flow statistics collection, see "Viewing message flow accounting and statistics data collection parameters by using the mqsireportflowstats command" on page 2736.

For troubleshooting information that includes guidance on dealing with performance issues, see "Troubleshooting performance problems" on page 2766.

Message flow statistics and accounting data

Message flow statistics and accounting data can be collected to record performance and operating details of one or more message flows.

You can use message flow statistics and accounting data to capture dynamic information about the runtime behavior of a message flow. For example, it indicates how many messages are processed and how large those messages are, processor usage and elapsed processing times. The statistical data is collected and recorded in a specified location when an event occurs. For example, when a snapshot interval expires or when the integration server that you are collecting information about stops. You can configure the data to be published in JSON or XML formats, or to be written in Comma Separated Values (CSV) format so that you can use it in a spreadsheet.

You can publish statistics and accounting data to one or more of the following destinations:

- The App Connect Enterprise web user interface (independent integration servers and integration servers that are managed by an integration node)
- Files
- User trace
- IBM MQ
- MQTT

You can use the statistics that are generated for the following purposes:

- You can use snapshot data to assess the execution of a message flow to determine why it, or a node within it, is not performing as you expect.
- You can determine the route that messages are taking through a message flow. For example, you might find that an error path is taken more frequently than you expect. You can use the statistics to understand when the messages are routed to this error path.

  Check the information provided by snapshot data for routing information. If this information is insufficient for your needs, use archive data.
- You can record the load that applications, business partners, or other users put on the integration server. You can use this information to calculate what to charge users based on their relative use of the integration server. For example, you might levy a nominal charge on every message that is processed by an integration server, or by a specific message flow. You can use archive data to make an assessment of this kind.

The integration server takes information about statistics and accounting from the operating system. On some operating systems, such as Windows and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

Data relating to the size of messages is not collected for WebSphere Adapters nodes. For example:

- The SAPInput node
- The FileInput node
- The JMSInput node
- Any user-defined input node that does not create a message tree from a bit stream.

You can start and stop data collection by modifying the statistics collection properties in the `node.conf.yaml` or `server.conf.yaml` configuration file for the integration node or server.

When an integration node or server is created, the publication of snapshot message flow statistics is enabled by default. The **`publicationOn`** property in the `node.conf.yaml` or `server.conf.yaml` file is explicitly set to `active` and the **`outputFormat`** property is set to `json`, which enables the publication of snapshot statistics to the web user interface. The publication of archive statistics is turned off by default.

For integration nodes and servers that were created prior to IBM App Connect Enterprise Version 11.0.0.8, the publication of all message flow statistics (snapshot and archive) was turned off by default. To enable the publication of statistics for integration nodes or servers that were created prior to V11.0.0.8, edit the relevant `.conf.yaml` file and activate the **`publicationOn`** and **`outputFormat`** properties.

For more information, see "Configuring the collection of message flow statistics by using a .yaml configuration file" on page 2729 and "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

You can activate data collection on your production and test systems. If you collect the default level of statistics (message flow), the effect on performance is minimal. However, collecting more data than the default message flow statistics can generate high volumes of report data that might affect performance slightly.

The information in the following topics helps you when you are planning your data collection:

- "Message flow accounting and statistics collection options" on page 2702
- "Message flow accounting and statistics accounting origin" on page 2702
- "Output formats for message flow accounting and statistics data" on page 2704

The following topic contains reference information that you might find helpful when you analyze and tune the performance of your message flows: "Message flow accounting and statistics records" on page 2703.

Publishing data to IBM MQ and MQTT

If you want to publish data to IBM MQ or MQTT, ensure that the publication of events is enabled and that the pub/sub broker is configured before you start data collection. For more information, see "Configuring the publication of event messages" on page 2807 and "Configuring the built-in MQTT pub/sub broker" on page 2811.

The topic for each message has the following structure:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/integrationNodeName/StatisticsAccounting/integration_server_name
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/integrationNodeName/StatisticsAccounting/integration_server_name
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/integrationNodeName/Statistics/JSON/integration_server_name
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/integrationNodeName/Statistics/JSON/integration_server_name
    ```

You can set up subscriptions for a specific integration server on a specific integration node. For example:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/INODE/StatisticsAccounting/default
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/INODE/StatisticsAccounting/default
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/INODE/Statistics/JSON/default
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/INODE/Statistics/JSON/default
    ```

You can also use wildcards in the subscriptions to broaden the scope of what is returned. For example, to subscribe to reports for all integration servers on all integration nodes, use the following values:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/+/StatisticsAccounting/#
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/+/StatisticsAccounting/#
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/+/Statistics/JSON/#
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/+/Statistics/JSON/#
    ```

Message flow accounting and statistics collection options

The options that you specify for message flow accounting and statistics collection determine what information is collected. You can configure the collection of snapshot data, archive data, or both.

- Snapshot data is collected for an interval of approximately 20 seconds. The exact length of the interval depends on system loading and the level of current integration server activity. You cannot modify the length of time for which snapshot data is collected. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.
- Archive data is collected for an interval that you have set for the integration node or integration server, by setting the **`Statistics.Archive.majorInterval`** property in the relevant `node.conf.yaml` or `server.conf.yaml` configuration file. You can specify an interval in the range 1 through 43200 minutes; the default value is 60 minutes. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.

  An interval ends and restarts when any of the following events occur, to preserve the integrity of the data that has been collected prior to the event:

  - The message flow is redeployed
  - The set of statistics data to be collected is modified
  - The integration node is shut down

You can modify the statistics properties in the `node.conf.yaml` or `server.conf.yaml` files to specify snapshot data collection, archive data collection, or both. When you have updated the `.yaml` file, you must restart the integration node or server for the changes to take effect. The state of the statistics collection that is configured in the `.yaml` file is applied to all flows that are deployed when the integration server starts, and also to any additional flows that are deployed afterwards. For information about configuring an integration server or integration node by editing a `.yaml` file, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172 and “Configuring an integration node by modifying the node.conf.yaml file” on page 118. Alternatively, by using the **`mqsichangeflowstats`** command, you can specify snapshot data collection, archive data collection, or both. You can activate snapshot data collection while archive data collection is active. The data recorded in both reports is the same, but is collected for different intervals. If you activate both snapshot and archive data collection, be careful not to combine information from the two different reports, because you might count information twice.

You can view statistics data as it is generated by selecting the **Statistics** tab in the web user interface.

For information about how to configure the collection of snapshot or archive statistics, see “Configuring the collection of message flow accounting and statistics data” on page 2728.

Message flow accounting and statistics accounting origin

Accounting and statistics data can be accumulated and reported with reference to an identifier associated with a message in a message flow. This identifier is the accounting origin, which provides a method of producing individual accounting and statistics data for multiple accounting origins that generate input to message flows. The accounting origin can be a fixed value, or it can be dynamically set according to your criteria.

For example, if your integration node hosts a set of message flows associated with a particular client in a single integration server, you can set a specific value for the accounting origin for all these flows. You can then analyze the output provided to assess the use that the client or department makes of the integration node, and charge them accordingly.

If you want to track the behavior of a particular message flow, you can set a unique accounting origin for this message flow, and analyze its activity over a specified period.

To make use of the accounting origin, you must perform the following tasks:

- Activate data collection, and request basic support by setting the **accountingOrigin** parameter in the `node.conf.yaml` or `server.conf.yaml` configuration file to `basic` (the default is none, or no support). For more information about configuring statistics collection, see "Configuring the collection of message flow accounting and statistics data" on page 2728.
- Configure each message flow for which you want a specific origin to include ESQL statements that set the unique value that is to be associated with the data collected. Data for message flows for which a specific value has not been set are identified with the value `Anonymous`.

  The ESQL statements can be coded in a Compute, Database, or Filter node.

  You can configure the message flow either to set a fixed value, or to determine a dynamic value, and can therefore create a very flexible method of recording sets of data that are specific to particular messages or circumstances. For more information, refer to "Setting message flow accounting and statistics accounting origin" on page 2735.

You can complete these tasks in either order; if you configure the message flow before starting data collection, the integration node ignores the setting. If you start data collection, specifying accounting origin support, before configuring the message flow, all data is collected with the Accounting Origin set to `Anonymous`. The integration node acknowledges the origin when you redeploy the message flow. You can also modify data collection that has already started to request accounting origin support from the time that you issue the command. In both cases, data that has already been collected is written out and collection is restarted.

When data has been collected, you can review information for one or more specific origins. For example, if you select XML publication messages as your output format, you can start an application that subscribes to the origin in which you are interested.

Message flow accounting and statistics records

You can collect message flow, thread, node, and terminal statistics for message flows.

Message flow statistics

One record is created for each message flow in an integration server. Each record contains the following details:

- Message flow name
- Integration server name
- Start and end times for data collection
- Type of data collected (snapshot or archive)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes
- Number of threads available and maximum assigned at any time
- Number of messages committed and backed out
- Accounting origin

Thread statistics

One record is created for each thread assigned to the message flow. Each record contains the following details:

- Thread number (this has no significance and is for identification only)
- Processor and elapsed time spent processing messages
- Processor and elapsed time spent waiting for input

- Number of messages processed
- Minimum, maximum, and average message sizes

Node statistics
One record is created for each node in the message flow. Each record contains the following details:

- Node name
- Node type (for example MQInput)
- Processor time spent processing messages
- Elapsed time spent processing messages
- Number of times that the node is invoked
- Number of messages processed
- Minimum, maximum, and average message sizes. When the node type is FileInput, the message size is reported as 0 because the size is not known at the time that the message is propagated.

Terminal statistics
One record is created for each terminal on a node. Each record contains the following details:

- Terminal name
- Terminal type (input or output)
- Number of times that a message is propagated to this terminal

For further details about specific output formats, see the following topics:

- "User trace entries for message flow accounting and statistics data" on page 2708
- "XML publication for message flow accounting and statistics data" on page 2718

Output formats for message flow accounting and statistics data
When you collect message flow statistics, you can choose the output destination for the data.

You can select one or more of the following destinations, by setting the **outputFormat** property in the configuration file for your integration node (`node.conf.yaml`) or integration server (`server.conf.yaml`):

- User trace
- "JSON publication" on page 2706
- XML publication
- CSV records

If no format is specified, accounting and statistics data is sent to the user trace log by default.

You can change the output format and destination of statistics data (snapshot, archive, or both) by setting the **outputFormat** property in the configuration file for your integration node (`node.conf.yaml`) or integration server (`server.conf.yaml`. You can set the output format of **snapshot** statistics data to one or more of the following values (separated by commas):

- csv
- json
- xml
- usertrace

You can set the output format of **archive** statistics data to one or more of the following values (separated by commas):

- csv
- xml
- usertrace

For more information about configuring the collection and publishing of message flow accounting and statistics data, see “Configuring the collection of message flow accounting and statistics data” on page 2728.

Before message flow accounting and statistics can be collected, you must ensure that the publication of events has been enabled and a pub/sub broker has been configured. For more information, see “Configuring the publication of event messages” on page 2807 and “Configuring the built-in MQTT pub/sub broker” on page 2811.

If you start the collection of message flow statistics data by using the web user interface, the statistics are emitted in JSON format in addition to any other formats that are already being emitted. If the output format was previously not specified and therefore defaulted to the user trace, the newly specified format replaces the default, and the data is no longer emitted to the user trace. However, if user trace has been explicitly specified, any additional formats that are selected subsequently are emitted in addition to the user trace.

If you use the **mqsichangeflowstats** command to explicitly specify the required output formats, the formats specified by the command replace the formats that are currently being emitted for the message flow (they are not added to them).

If you stop statistics collection from the web user interface, all output formats are turned off. If statistics collection is subsequently restarted by using the **mqsichangeflowstats** command, the output format is reset to the default value of `user trace`, unless other formats are specified on the command. However, if statistics collection is restarted by using the web user interface, data is collected in JSON format.

Statistics data is written to the specified output location in the following circumstances:

- When the archive data interval expires.
- When the snapshot interval expires.
- When the integration node shuts down. Any data that has been collected by the integration node, but has not yet been written to the specified output destination, is written during shutdown. It might therefore represent data for an incomplete interval.
- When any part of the integration node configuration is redeployed. Redeployed configuration data might contain an updated configuration that is not consistent with the existing record structure (for example, a message flow might include an additional node, or an integration server might include a new message flow). Therefore the current data, which might represent an incomplete interval, is written to the output destination. Data collection continues for the redeployed configuration until you change data collection parameters or stop data collection.
- When data collection parameters are modified. If you update the parameters that you have set for data collection, all data that is collected for the message flow (or message flows) is written to the output destination to retain data integrity. Statistics collection is restarted according to the new parameters.
- When an error occurs that terminates data collection. You must restart data collection yourself in this case.

User trace entries

You can specify that the data that is collected is written to the user trace log. The data is written even when trace is switched off.

If no output destination is specified for accounting and statistics, the default is the user trace log. If one or more output formats are subsequently specified, the specified formats replace the default, and the data is no longer emitted to the user trace. However, if user trace has been explicitly specified, any additional formats that are selected subsequently are emitted in addition to the user trace.

The data is written to one of the following locations:

 **Windows**

If you set the work path by using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\Common\log`.

If you have not specified the integration node work path, the location is:

- On Windows:`C:\ProgramData\IBM\MQSI\Common\log`.

Linux UNIX **Linux**

`/var/mqsi/common/log`

For information about the user trace entries, see "User trace entries for message flow accounting and statistics data" on page 2708.

JSON publication

You can specify that the data that is collected is published in JSON format, which is available for viewing in the web user interface. If statistics collection is started through the web user interface, statistics data is emitted in JSON format in addition to any other formats that are already being emitted.

The topic on which the data is published has the following structure:

- For publications on an MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/Statistics/JSON/SnapShot/integrationServerName/
applications/application_name
/libraries/library_name/messageflows/message_flow_name
```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/Statistics/JSON/SnapShot/integrationServerName/
applications/application_name
/libraries/library_name/messageflows/message_flow_name
```

The variables correspond to the following values:

integrationNodeName
The name of the integration node for which statistics are collected. For an integration server that is managed by an integration node, *integrationNodeName* is the name of the integration node that manages the integration server. For an independent integration server, specify the literal string `integration_server` in place of an integration node name.

integration_server_name
The name of the integration server for which statistics are collected

application_name
The name of the application for which statistics are collected

library_name
The name of the library for which statistics are collected

message_flow_name
The name of the message flow for which statistics are collected

For information about the JSON publication, see "JSON publication for message flow accounting and statistics data" on page 2713.

XML publication

You can specify that the data that is collected is published in XML format and is available to subscribers registered in the integration node network that subscribe to the correct topic.

The topic on which the data is published has the following structure:

- For publications on an MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/StatisticsAccounting/record_type/integrationServerName/message_flow_label
```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/
StatisticsAccounting/record_type/integrationServerName/message_flow_label
```

The variables correspond to the following values:

integrationNodeName
The name of the integration node for which statistics are collected. For an integration server that is managed by an integration node, *integrationNodeName* is the name of the integration node that manages the integration server. For an independent integration server, specify the literal string `integration_server` in place of an integration node name.

record_type
Set to `SnapShot` or `Archive`, depending on the type of data to which you are subscribing. Alternatively, use *+* to register for both snapshot and archive data if it is being produced. This value is case sensitive and must be entered as `SnapShot`.

integrationServerName
The name of the integration server for which statistics are collected.

message_flow_label
The label on the message flow for which statistics are collected.

Subscribers can include filter expressions to limit the publications that they receive. For example, they can choose to see only snapshot data, or to see data that is collected for a single integration node. Subscribers can specify wild cards (*+* and *#*) to receive publications that refer to multiple resources. Use *+* to receive resources on one topic level, and *#* to receive resources across multiple topic levels.

The following examples show the topic with which a subscriber registers to receive different sorts of data:

- Register the following topic for the subscriber to receive data for all message flows running on an integration node named INODE:

  ```
  $SYS/Broker/INODE/StatisticsAccounting/#
  ```

  or

  ```
  IBM/IntegrationBus/INODE/StatisticsAccounting/#
  ```

- Register the following topic for the subscriber to receive only archive statistics that relate to a message flow `Flow1` running on integration server `default` on integration node INODE:

  ```
  $SYS/Broker/INODE/StatisticsAccounting/Archive/default/Flow1
  ```

  or

  ```
  IBM/IntegrationBus/INODE/StatisticsAccounting/Archive/default/Flow1
  ```

- Register the following topic for the subscriber to receive both snapshot and archive data for message flow `Flow1` running on integration server `default` on integration node INODE:

  ```
  $SYS/Broker/INODE/StatisticsAccounting/+/default/Flow1
  ```

  or

  ```
  IBM/IntegrationBus/INODE/StatisticsAccounting/+/default/Flow1
  ```

For help with registering your subscriber, see Message display, test and performance utilities SupportPac (IH03).

For information about the XML publication, see “XML publication for message flow accounting and statistics data” on page 2718.

CSV records

You can specify that the data that is collected is published in comma-separated value (`.csv`) format. Snapshot and archive data records are written to output files, which include a header with the field name. The fields for averages are optional, and are written only if the `averages` property of the statistics file writer is set to `true`.

One line is written for each message flow that is producing data for the time period that you choose. For example, if `MessageFlowA` and `MessageFlowB` are both producing archive data over a period of 60 minutes, both message flows produce a line of statistics data every 60 minutes.

For more information about the CSV records, see "CSV file format for message flow accounting and statistics data" on page 2724.

User trace entries for message flow accounting and statistics data
Certain information is written to the user trace log for message flow accounting and statistics data.

The data records are identified by the following message numbers:

- BIP2380I
- BIP2381I
- BIP2382I
- BIP2383I

The inserts for each message are described in the following tables.

This table describes the inserts in message BIP2380I. One message is written for the message flow.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Type	Character	Type of output, one of: • Archive • Snapshot
Reason	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified
BrokerLabel	Character (maximum 32)	Integration node name
BrokerUUID	Character (maximum 32)	Integration node universal unique identifier
ExecutionGroupName	Character (maximum 32)	Integration server name
ExecutionGroupUUID	Character (maximum 32)	Integration server universal unique identifier

Field	Data type	Details
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
GMTStartTime	Character	Interval start date and time (universal timestamp in ISO8601)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
GMTEndTime	Character	Interval end date and time (universal timestamp in ISO8601)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum input message size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached

Field	Data type	Details
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors while processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts while processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the inserts in message BIP2381I. One message is written for each thread.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)

Field	Data type	Details
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the inserts in message BIP2382I. One message is written for each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the inserts in message BIP2383I. One message is written for each terminal on each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key that is used to associate related accounting and statistics BIP messages
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

For an example of user trace entries, see “Example of user trace entries for message flow accounting and statistics” on page 2712.

Example of user trace entries for message flow accounting and statistics
This example shows a user trace that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The messages that are written to the trace show that the message flow is called *myExampleFlow*, and that it is running in an integration server named *default* on integration node TESTNODE_*user_name* . The message flow contains the following message flow nodes:

- An MQInput node called *inNode*
- A Compute node called *First1*
- An MQOutput node called *outNode*

The nodes are connected together (Out terminal to In terminal for each connection).

During the interval for which statistics have been collected, this message flow processed 150 input messages.

The records show that two threads are assigned to this message flow. One thread is assigned when the message flow is deployed (the default number); an additional thread (thread 0) listens on the input queue. The listening thread starts additional threads to process input messages that are dependent on the number of instances that you have configured for the message flow, and on the rate of arrival of the input messages on the input queue.

The following command has been issued to achieve these results:

```
mqsichangeflowstats TESTNODE_user_name -s -c active -e default -f myExampleFlow -n advanced -t basic -b
 basic
```

The integration node takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

```
BIP2380I: IIB message flow statistics. ProcessID='328467', Key='6', Type='SnapShot', Reason='Snapshot',
IntegrationNodeLabel='TESTNODE_user_name', BrokerUUID='18792e66-e100-0000-0080-f197e5ed81bd',
IntegrationServerName='default', IntegrationServerUUID='15d4314a-3607-11d4-8000-09140f7b0000',
MessageFlowName='myExampleFlow',
StartDate='2003-05-20', StartTime='13:44:31.885862',
EndDate='2003-05-20', EndTime='13:44:51.310080',
TotalElapsedTime='9414843', MaximumElapsedTime='1143442', MinimumElapsedTime='35154',
TotalCPUTime='760147', MaximumCPUTime='70729', MinimumCPUTime='3124',
CPUTimeWaitingForInputMessage='45501', ElapsedTimeWaitingForInputMessage='11106438',
TotalInputMessages='150', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915',
NumberOfThreadsInPool='1', TimesMaximumNumberOfThreadsReached='150',
TotalNumberOfMQErrors='0', TotalNumberOfMessagesWithErrors='0',
TotalNumberOfErrorsProcessingMessages='0', TotalNumberOfTimeOuts='0',
TotalNumberOfCommits='150', TotalNumberOfBackouts='0', AccountingOrigin="DEPT2".
Statistical information for message flow 'myExampleFlow' in integration node 'MQ01BRK'.
This is an information message produced by IIB statistics.

BIP2381I: IIB thread statistics. ProcessID='328467', Key='6', Number='0',
TotalNumberOfInputMessages='0',
TotalElapsedTime='0', TotalCPUTime='0', CPUTimeWaitingForInputMessage='110',
ElapsedTimeWaitingForInputMessage='5000529', TotalSizeOfInputMessages='0',
MaximumSizeOfInputMessages='0', MinimumSizeOfInputMessages='0'.
Statistical information for thread '0'.
This is an information message produced by IIB statistics.

BIP2381I: IIB thread statistics. ProcessID='328467', Key='6', Number='18',
TotalNumberOfInputMessages='150',
TotalElapsedTime='9414843', TotalCPUTime='760147', CPUTimeWaitingForInputMessage='45391',
ElapsedTimeWaitingForInputMessage='6105909', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915'.
Statistical information for thread '18'.
This is an information message produced by IIB statistics.
```

```
BIP2382I: IIB node statistics. ProcessID='328467', Key='6',
Label='First1', Type='ComputeNode',
TotalElapsedTime='6428815', MaximumElapsedTime='138261', MinimumElapsedTime='28367',
TotalCPUTime='604060', MaximumCPUTime='69645', MinimumCPUTime='2115',
```

```
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'First1'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by IIB statistics.
```

```
BIP2382I: IIB node statistics. ProcessID='328467', Key='6',
Label='inNode', Type='MQInputNode',
TotalElapsedTime='1813446', MaximumElapsedTime='1040209', MinimumElapsedTime='1767',
TotalCPUTime='70565', MaximumCPUTime='686', MinimumCPUTime='451',
CountOfInvocations='150', NumberOfInputTerminals='0', NumberOfOutputTerminals='3'.
Statistical information for node 'inNode'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='catch', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'catch'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by IIB statistics.

BIP2382I: IIB node statistics. ProcessID='328467', Key='6',
Label='outNode', Type='MQOutputNode',
TotalElapsedTime='1172582', MaximumElapsedTime='177516', MinimumElapsedTime='3339',
TotalCPUTime='85522', MaximumCPUTime='762', MinimumCPUTime='536',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'outNode'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by IIB statistics.

BIP2383I: IIB terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'out'.
This is an information message produced by IIB statistics.
```

JSON publication for message flow accounting and statistics data
Information is written to the JSON publication for message flow accounting and statistics data.

The data is created in the folder `WMQIStatisticsAccounting`, which contains subfolders that provide more detailed information. All folders are present in the publication even if you set current data collection parameters to specify that the relevant data is not collected. The data is published to the following topics:

Snapshot data

- For publications on an MQ pub/sub broker:

  ```
  $SYS/Broker/integrationNodeName/Statistics/JSON/SnapShot/
  ```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/Statistics/JSON/SnapShot/
```

Archive data

- For publications on an MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/Statistics/JSON/Archive/
```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/Statistics/JSON/Archive/
```

For an integration server that is managed by an integration node, *integrationNodeName* is the name of the integration node that manages the integration server. For an independent integration server, specify the literal string `integration_server` in place of an integration node name.

Snapshot data is used for performance analysis, and is published as retained and not persistent. Snapshot data is produced every 20 seconds, and you can view it in the web user interface to help you analyze the performance of your message flows. If statistics collection is started through the web user interface, statistics data is emitted in JSON format in addition to any other formats that are already being emitted.

Archive data is used for accounting where an audit trail might be required, and is published as retained and persistent. All publications are global and can be collected by a subscriber that is registered anywhere in the network. They can also be collected by more than one subscriber.

One JSON publication is generated for each message flow that is producing data for the time period that you choose. For example, if MessageFlowA and MessageFlowB are both producing archive data over a period of 60 minutes, both MessageFlowA and MessageFlowB produce a JSON publication every 60 minutes.

If you are concerned about the safe delivery of these messages (for example, for charging purposes), use a secure delivery mechanism such as IBM MQ.

The folders and subfolders in the JSON publication have the following identifiers:

- WMQIStatisticsAccounting
- MessageFlow
- ThreadStatistics
- NodesStatistics
- TerminalStatistics

The tables provided here describe the contents of each of these folders.

The following table describes the general accounting and statistics information, created in folder WMQIStatisticsAccounting.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

Field	Data type	Details
NumberOfThreads	Numeric	Number of thread statistics subfolders in WMQIStatisticsAccounting folder
NumberOfNodes	Numeric	Number of node statistics subfolders in WMQIStatisticsAccounting folder

The following table describes the message flow statistics information, created in folder MessageFlow.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Integration node name
BrokerUUID	Character (maximum 32)	Integration node universal unique identifier
ExecutionGroupName	Character (maximum 32)	Integration server name
ExecutionGroupUUID	Character (maximum 32)	Integration server universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
GMTStartTime	Character	Interval start date and time (universal timestamp in ISO8601)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
GMTEndTime	Character	Interval end date and time (universal timestamp in ISO8601)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)

Field	Data type	Details
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue, but before it propagates it to the output terminal (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors when processing a message

Field	Data type	Details
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts when processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the thread statistics information for each individual thread, created in folder ThreadStatistics.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the node statistics information for each individual node, created in folder NodesStatistics.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)

Field	Data type	Details
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the terminal statistics information, created in folder TerminalStatistics.

Field	Data type	Details
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

XML publication for message flow accounting and statistics data
Certain information is written to the XML publication for message flow accounting and statistics data.

The data is created in the folder `WMQIStatisticsAccounting`, which contains subfolders that provide more detailed information. All folders are present in the publication even if you set current data collection parameters to specify that the relevant data is not collected.

Snapshot data is used for performance analysis, and is published as retained and not persistent. Archive data is used for accounting where an audit trail might be required, and is published as retained and persistent. All publications are global and can be collected by a subscriber that is registered anywhere in the network. They can also be collected by more than one subscriber.

One XML publication is generated for each message flow that is producing data for the time period that you choose. For example, if MessageFlowA and MessageFlowB are both producing archive data over a period of 60 minutes, both MessageFlowA and MessageFlowB produce an XML publication every 60 minutes.

If you are concerned about the safe delivery of these messages (for example, for charging purposes), use a secure delivery mechanism such as IBM MQ.

The folders and subfolders in the XML publication have the following identifiers:

- WMQIStatisticsAccounting
- MessageFlow
- Threads
- ThreadStatistics
- Nodes
- NodesStatistics
- TerminalStatistics

The tables provided here describe the contents of each of these folders.

The following table describes the general accounting and statistics information, created in folder WMQIStatisticsAccounting.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

The following table describes the message flow statistics information, created in folder MessageFlow.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Integration node name. For an independent integration server, it is `integration_server`.
BrokerUUID	Character (maximum 32)	Integration node universal unique identifier (not used in IBM App Connect Enterprise Version 11.0)
ExecutionGroupName	Character (maximum 32)	The value set on the **--name** parameter of the **IntegrationServer** command.
ExecutionGroupUUID	Character (maximum 32)	Integration server universal unique identifier (not used in IBM App Connect Enterprise Version 11.0)
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
GMTStartTime	Character	Interval start date and time (universal timestamp in ISO8601)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
GMTEndTime	Character	Interval end date and time (universal timestamp in ISO8601)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)

Field	Data type	Details
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.

Field	Data type	Details
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue, but before it propagates it to the output terminal (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors when processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts when processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the thread statistics information, created in folder `Threads`.

Field	Data type	Details
Number	Numeric	Number of thread statistics subfolders in Threads folder

The following table describes the thread statistics information for each individual thread, created in folder ThreadStatistics, a subfolder of Threads.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)

Field	Data type	Details
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the node statistics information, created in folder Nodes.

Field	Data type	Details
Number	Numeric	Number of node statistics subfolders in Nodes folder

The following table describes the node statistics information for each individual node, created in folder NodesStatistics, a subfolder of Nodes.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the terminal statistics information, created in folder TerminalStatistics.

Field	Data type	Details
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

For an example of an XML publication, see "Example of an XML publication for message flow accounting and statistics" on page 2723.

Example of an XML publication for message flow accounting and statistics
This example shows an XML publication that contains message flow accounting and statistics data.

The following example shows what is generated for a snapshot report. The content of this publication message shows that the message flow is called *XMLflow*, and that it is running in an integration server that is named *default* on integration node *MQ02BRK*. The message flow contains the following nodes:

- An MQInput node called *INQueue3*.
- An MQOutput node called *OUTQueue*.
- An MQOutput node called *FAILQueue*.

The MQInput node's Out terminal is connected to the OUTQueue node. The MQInput node's Failure terminal is connected to the FAILQueue node.

During the interval for which statistics are collected, this message flow processed no messages.

A publication that is generated for this data always includes the appropriate folders, even if there is no current data.

Blank lines are added between folders to improve readability.

The following command was run to achieve these results:

```
mqsichangeflowstats MQ02BRK -s -c active -e default -k XMLApp -f XMLFlow -n advanced -t basic -b basic -
o xml
```

The integration node takes information about statistics and accounting from the operating system. On some operating systems, such as Windows, UNIX, and Linux, rounding can occur because the system calls that are used to determine the processor times are not sufficiently granular. This rounding might affect the accuracy of the data.

The following example is the subscription message. The <psc> and <mcd> elements are part of the RFH header.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>IBM/$SYSBroker/MQ02BRK/StatisticsAccounting/SnapShot/default/XMLflow
  </Topic>
</psc>


<mcd>
  <Msd>xml</Msd>
</mcd>
```

The following example is the publication that the integration node generates:

```
<WMQIStatisticsAccounting RecordType="SnapShot" RecordCode="Snapshot">

<MessageFlow BrokerLabel="MQ02BRK"
 BrokerUUID="7d951e31-f200-0000-0080-efe1b9d849dc"
 ExecutionGroupName="default"
 ExecutionGroupUUID="77cf1e31-f200-0000-0080-efe1b9d849dc"
 MessageFlowName="XMLflow" StartDate="2003-01-17"
 StartTime="14:44:34.581320" EndDate="2003-01-17" EndTime="14:44:44.582926"
 TotalElapsedTime="0"
 MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
 MaximumCPUTime="0"MinimumCPUTime="0" CPUTimeWaitingForInputMessage="685"
 ElapsedTimeWaitingForInputMessage="10001425" TotalInputMessages="0"
 TotalSizeOfInputMessages="0" MaximumSizeOfInputMessages="0"
 MinimumSizeOfInputMessages="0" NumberOfThreadsInPool="1"
 TimesMaximumNumberOfThreadsReached="0" TotalNumberOfMQErrors="0"
 TotalNumberOfMessagesWithErrors="0" TotalNumberOfErrorsProcessingMessages="0"
 TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages="0"
 TotalNumberOfCommits="0" TotalNumberOfBackouts="0" AccoutingOrigin="DEPT1"/>


<Threads Number="1">
<ThreadStatistics Number="5" TotalNumberOfInputMessages="0"
TotalElapsedTime="0" TotalCPUTime="0" CPUTimeWaitingForInputMessage="685"
ElapsedTimeWaitingForInputMessage="10001425" TotalSizeOfInputMessages="0"
```

```
MaximumSizeOfInputMessages="0" MinimumSizeOfInputMessages="0"/>
</Threads>


<Nodes Number="3">


  <NodeStatistics Label="FAILQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="INQueue3" Type="MQInput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0"CountOfInvocations="0"
    NumberOfInputTerminals="0" NumberOfOutputTerminals="3">
   <TerminalStatistics Label="catch" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="OUTQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
    <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
    <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
    <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
   </NodeStatistics>


 </Nodes>


</WMQIStatisticsAccounting>
```

CSV file format for message flow accounting and statistics data

Snapshot and archive data records are written to output files in comma-separated value (`.csv`) format.

The output files include a header with the field name. The fields for averages are optional, and are written only if the `averages` property of the statistics file writer is set to `true`. One line is written for each message flow that is producing data for the time period that you choose. For example, if `MessageFlowA` and `MessageFlowB` are both producing archive data over a period of 60 minutes, both message flows produce a line of statistics data every 60 minutes.

The following table describes the general accounting and statistics information. These fields appear in all file types.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

The following table describes the message flow statistics information, which is written to the `flowStats` files.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Integration node name
BrokerUUID	Character (maximum 32)	Integration node universal unique identifier
ExecutionGroupName	Character (maximum 32)	Integration server name
ExecutionGroupUUID	Character (maximum 32)	Integration server universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
GMTStartTime	Character	Interval start date and time (universal timestamp in ISO8601)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
GMTEndTime	Character	Interval end date and time (universal timestamp in ISO8601)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
AverageElapsedTime	Numeric	Average elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time that is spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time that is spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)

Field	Data type	Details
AverageCPUTime	Numeric	Average processor time that is spent processing an input message (microseconds)
MaximumCPUTime	Numeric	Maximum processor time that is spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed TotalInputMessages records only those messages that are propagated from input node terminals.
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
AverageSizeOfInputMessages	Numeric	Average size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors	Numeric	Number of MQGET errors (MQInput node) or web services errors (HTTPInput node) For example, a conversion error occurs when the message is got from the queue.
TotalNumberOfMessagesWithErrors	Numeric	Number of messages that contain errors These errors include exceptions that are thrown downstream of the input node, and errors that are detected by the input node after it successfully retrieves the message from the queue, but before it propagates it to the output terminal (for example, a format error). TotalNumberOfMessagesWithErrors can include messages that are not included in TotalInputMessages.

Field	Data type	Details
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors when processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts when processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

The following table describes the thread statistics information, created in `threadStats` files.

Field	Data type	Details
Number	Numeric	Number of thread statistics subfolders in Threads folder

The following table describes the thread statistics information for each individual thread, created in `threadStats` files.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages that are processed by a thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
AverageElapsedTime	Numeric	Average elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
AverageCPUTime	Numeric	Average processor time that is spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total processor time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time that is spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
AverageSizeOfInputMessages	Numeric	Average size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the node statistics information, created in `nodeStats` files.

Field	Data type	Details
Number	Numeric	Number of node statistics subfolders in Nodes folder

The following table describes the node statistics information for each individual node, created in `nodeStats` files.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
AverageElapsedTime	Numeric	Average elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total processor time spent processing input messages (microseconds)
AverageCPUTime	Numeric	Average processor time that is spent processing an input message (microseconds)
MaximumCPUTime	Numeric	Maximum processor time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum processor time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages that are processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

Configuring the collection of message flow accounting and statistics data

You can configure the collection of message flow statistics and accounting data by modifying properties in .yaml configuration files, or by running the **`mqsichangeflowstats`** command.

Before you begin

- Read the concept topic "Message flow statistics and accounting data" on page 2699

About this task

You can configure message flow accounting and statistics data collection, either by modifying properties in the `server.conf.yaml` or `node.conf.yaml` configuration files, or by running the command. You can configure the collection of snapshot data, archive data, or both.

Procedure

Configure the collection of message flow accounting and statistics data by completing the steps in either of the following tasks:

- "Configuring the collection of message flow statistics by using a .yaml configuration file" on page 2729
- "Configuring the collection of message flow statistics by using the mqsichangeflowstats command" on page 2732

Configuring the collection of message flow statistics by using a .yaml configuration file

Configure the collection of accounting and statistics data for one or more active message flows by setting the statistics properties in the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or integration server.

Before you begin

- Read the topics "Message flow statistics and accounting data" on page 2699 and "Configuring the collection of message flow accounting and statistics data" on page 2728.
- Configure an integration server, by following the instructions described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
- Create your message flow and deploy it to your integration server. For more information, see "Creating a message flow" on page 574 and "Deploying integration solutions to a production environment" on page 2480.
- Ensure that the publication of events is enabled, and the pub/sub broker is correctly configured. For an integration node, the default configuration enables publication of events by using the built-in MQTT pub/sub broker. If IBM MQ is installed and there is a default IBM MQ queue manager that is specified on the integration node or independent integration server, the IBM MQ pub/sub broker is used for the publication of events. For more information, see "Configuring the publication of event messages" on page 2807 and "Configuring the built-in MQTT pub/sub broker" on page 2811.

About this task

You can configure message flow accounting and statistics data collection by modifying properties in the `server.conf.yaml` or `node.conf.yaml` files. For example, you can start collecting data for a new message flow that you deploy to an integration server for which you are already collecting data. When you restart the integration server, the new settings that you define in the `.yaml` configuration file come into effect.

You can configure the collection of snapshot data, archive data, or both, by setting properties in the Statistics section of the configuration file, as shown in the following example:

```
Statistics:
# All applications and message flows will inherit the Snapshot and Archive values set here,
 unless they have been set
# to a specific value other than inherit via the WebUI, mqsichangeflowstats command, Toolkit or
 apiv2 REST
# Notes
#  - values here can be overridden by 'overrides/server.conf.yaml'
#  - to publish on MQ or MQTT, also configure Events.OperationalEvents, and set outputFormat to
 include json and/or xml
#  - to display in the WebUI Snapshot.outputFormat must include json; nodeDataLevel needs to be
 set to basic or advanced

  Snapshot:
     publicationOn: 'active'     # Choose 1 of : active|inactive. Explictly set to 'active' by
 default. If unset, defaults to 'inactive'.
                                 # Also set Events.OperationalEvents.MQ|MQTT for outputFormat
 json,xml to be published to MQ/MQTT
    #accountingOrigin: 'none'    # Choose 1 of : none|basic. Default is 'none'.
    #nodeDataLevel: 'basic'      # Choose 1 of : none|basic|advanced. Explicitly set to 'basic'
 by default. If unset, defaults to 'none'.
     outputFormat: 'json'        # Choose comma-separated list, one or more of :
 csv,json,xml,usertrace. Explicitly set to 'json' by default (for web UI). If unset, defaults to
 ''.
    #threadDataLevel: 'none'     # Choose 1 of : none|basic. If unset, defaults to 'none'.
  Archive:
     archivalOn: 'active'        # Choose 1 of : active|inactive. Default is 'inactive'.
                                 # Also set Events.OperationalEvents.MQ|MQTT for outputFormat
 xml to be published to MQ/MQTT
    #accountingOrigin: 'none'    # Choose 1 of : none|basic
    #majorInterval: 60           # Sets the interval in minutes at which archive statistics are
 published
    #nodeDataLevel: 'none'       # Choose 1 of : none|basic|advanced
     outputFormat: 'usertrace'   # Comma-separated list of : csv,xml,usertrace
    #threadDataLevel: 'none'     # Choose 1 of : none|basic
```

Alternatively, you can configure the collection of message flow accounting and statistics data dynamically, by running the command. The settings that are specified by the command take effect immediately, without restarting the integration server. For more information, see "Modifying message flow accounting and statistics data collection settings by using the mqsichangeflowstats command" on page 2737.

Procedure

Complete the following steps to configure the collection of accounting and statistics data for message flows by uncommenting and setting the statistics properties in the configuration file for your integration node or server. This configuration is then applied to all message flows when they are deployed or when the integration server is restarted:

1. Open the appropriate `node.conf.yaml` or `server.conf.yaml` configuration file by using a YAML editor.

   If you are not able to access a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters. Tab characters are not valid in YAML files and they cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

You can start collection of snapshot data, archive data, or both, by setting properties in the Statistics section of the configuration file (`node.conf.yaml` or `server.conf.yaml`):

2. To turn on the collection of **snapshot** statistics data, uncomment and set the following properties in the appropriate `.yaml` file:
   a) Enable the collection of snapshot statistics data by setting the **publicationOn** property to `active`. This setting takes effect when the integration server is restarted. However, it can be changed dynamically (without restarting the integration server) by using the command.

      When a `node.conf.yaml` or `server.conf.yaml` file is created, the **publicationOn** property is explicitly set to `active` by default, and snapshot statistics for the integration node or server are published to the web user interface. If the **publicationOn** property is unset (with a value of `''`), the publication of snapshot statistics is turned off.

      In `node.conf.yaml` and `server.conf.yaml` files that were created prior to IBM App Connect Enterprise Version 11.0.0.8, the **publicationOn** property was set to `inactive` by default, so the publication of statistics was turned off. To enable the publication of snapshot statistics for those integration nodes or servers, edit the relevant `.conf.yaml` file and set the **publicationOn** property to `active`.

   b) Decide the target destination and specify the appropriate format in the **outputFormat** property.

      If you want to display statistics in the web user interface, you must specify `json` as one of the values in the **outputFormat** property. The web user interface can display message flow statistics

and accounting data for independent integration servers and integration servers that are managed by an integration node. The data that it receives must be in JSON format.

When a `node.conf.yaml` or `server.conf.yaml` file is created, the **outputFormat** property is explicitly set to `json` by default. If this property is unset, the output format is set to `''`.

You can choose multiple destinations by specifying multiple comma-separated values:

- `usertrace`

  The data is sent to the user trace log, which must be enabled before you can view the data.

- `csv`

  The data is sent to a set of cyclic files in the work directory of the integration node or integration server.

- `json`

  The data is sent to the IBM App Connect Enterprise web user interface. If IBM MQ and MQTT are enabled, data is also sent there. When a `node.conf.yaml` or `server.conf.yaml` file is created, the **outputFormat** property is explicitly set to `json` by default.

- `xml`

  If IBM MQ and MQTT are enabled, the data is also sent there.

c) Optional: Decide whether you want to associate data collection with a particular accounting origin. To report an accounting origin for the data, set the **accountingOrigin** property to `basic`.

Accounting and statistics data can be accumulated and reported about an identifier that is associated with a message in a message flow. This identifier is the accounting origin, which provides a method of producing individual accounting and statistics data for multiple accounting origins that generate input to message flows. The default for this property is `none` (no support). For more information, see "Setting message flow accounting and statistics accounting origin" on page 2735.

d) Optional: Decide whether you want to collect thread-related statistics. To include thread-related data, set the **threadDataLevel** property to `basic`. The default value for this property is `none`.

e) Optional: Decide whether you want to collect node-related statistics. If you do, you can also collect information about terminals for the nodes. To include node-related data, set the **nodeDataLevel** property to `basic`, or to include node-related and terminal-related data, specify `advanced`.

When the collection of snapshot statistics data is turned on, an up-to-date snapshot of information is collected every 20 seconds. You can view the snapshot data in the web user interface. For more information, see "Viewing message flow statistics and accounting data" on page 2738.

3. To configure the collection of **archive statistics data**, uncomment and set the following properties in the appropriate `.yaml` file:
   a) Set the **`archivalOn`** property to `active`.

      This setting takes effect when the integration server is restarted or when the message flow is deployed. However, it can be changed dynamically (without restarting the integration server) by using the command.

   b) Set the **`outputFormat`** property to one or more of the following values (separated by commas and enclosed by single quotation marks):

      - `usertrace`
      - `csv`
      - `xml`

      By default, the publication of archive statistics data is turned off. If you start collection (by setting the **`archivalOn`** property to `active`), the default output format is `usertrace`.

   c) Optional: Specify the interval (in minutes) at which archive statistics are published, by setting the **`majorInterval`** property. The default is `60` minutes.

   d) Optional: To report an accounting origin for the data, set the **`accountingOrigin`** property to `basic`.

      Accounting and statistics data can be accumulated and reported about an identifier that is associated with a message in a message flow. This identifier is the accounting origin, which provides a method of producing individual accounting and statistics data for multiple accounting origins that generate input to message flows. The default for this property is `none` (no support). For more information, see "Message flow accounting and statistics accounting origin" on page 2702.

   e) Optional: To include thread-related data in the statistics, set the **`threadDataLevel`** property to `basic`. The default value for this property is `none`.

   f) Optional: To include node-related data in the statistics, set the **`nodeDataLevel`** property to `basic`, or to include node-related and terminal-related data in the statistics, specify `advanced`.

4. Restart the integration node or integration server for the changes to take effect. Changes to the `server.conf.yaml` and `node.conf.yaml` files take effect only when the modified integration node or server is restarted.

   The state of statistics collection that is configured in the `.yaml` file is applied to all flows that are deployed when the integration server starts. It is also applied to any additional flows that are deployed after the integration server starts.

Configuring the collection of message flow statistics by using the `mqsichangeflowstats` command

You can configure the collection of accounting and statistics data for your message flows by using the **`mqsichangeflowstats`** command.

Before you begin

- Read the topics "Message flow statistics and accounting data" on page 2699 and "Configuring the collection of message flow accounting and statistics data" on page 2728.
- Configure an integration server by following the instructions described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
- Create your message flow and deploy it to your integration server. For more information, see "Creating a message flow" on page 574 and "Deploying integration solutions to a production environment" on page 2480.
- Ensure that the publication of events is enabled, and the pub/sub broker is correctly configured. For an integration node, the default configuration enables publication of events that use the built-in MQTT pub/sub broker. If IBM MQ is installed and there is a default IBM MQ queue manager that is specified on the integration node or independent integration server, the IBM MQ pub/sub broker is used for the

publication of events. For more information, see “Configuring the publication of event messages” on page 2807 and “Configuring the built-in MQTT pub/sub broker” on page 2811.

About this task

You can configure the collection of message flow accounting and statistics data for one or more active message flows by using the command.

Alternatively, you can configure the statistics and accounting properties in the `.yaml` configuration file for your integration node or server, as described in “Configuring the collection of message flow statistics by using a .yaml configuration file” on page 2729.

If you want to view message flow statistics by using the web user interface, you must specify JSON as one of the output formats when you configure statistics collection. You can also report the statistics to the user trace, IBM MQ, or MQTT.

Procedure

If you use the **mqsichangeflowstats** command to start statistics collection and specify the required output formats, the specified formats replace the output formats that were previously emitted for the message flow. The specified formats are not added to the previously emitted formats. If you start statistics collection by running the **mqsichangeflowstats** command without specifying an output format on the command, the statistics data is emitted to the user trace log by default.

Select the granularity of the data that you want to be collected by specifying the appropriate parameters on the **mqsichangeflowstats** command.

To configure the collection of message flow accounting and statistics data, complete the following steps:

1. Identify the resource for which you want to collect statistics.

   You can collect statistics for a specific message flow, for all message flows in a specified application or library, or for all message flows on a specified integration server.
2. Decide whether you want to collect thread-related statistics.
3. Decide whether you want to collect node-related statistics.

   If you do, you can also collect information about terminals for the nodes.
4. Decide whether you want to associate data collection with a particular accounting origin.

   This option is valid for snapshot and archive data, and for message flows and integration servers. However, when active, you must set its origin value in each message flow to which it refers. If you do not configure the participating message flows to set the appropriate origin identifier, the data that is collected for that message flow is collected with the origin set to `Anonymous`.

   For more information, see “Setting message flow accounting and statistics accounting origin” on page 2735.
5. Decide the target destination. You can choose multiple destinations by specifying multiple comma-separated values for the output format property.
   - User trace log (the default setting).
   - XML format publication message. If you chose this target destination, register the following topic for the subscriber:
     - For publications on an IBM MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/StatisticsAccounting/recordType/integrationServerName/messageFlowLabel
```

     - For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/
StatisticsAccounting/recordType/integrationServerName/messageFlowLabel
```

       where *integrationNodeName*, *integrationServerName*, and *messageFlowLabel* represent the integration node, integration server, and message flow on which you want to receive data. *recordType*

is the type of data collection on which you want to receive publications (SnapShot, Archive, or the topic wildcard "+" to receive both). The value that you specify for record type is case-sensitive, so if you choose to receive snapshot data, set the record type to `SnapShot`.

- JSON publication message. Register the following topic for the subscriber:
  - For publications on an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/integrationNodeName/Statistics/JSON/recordType/integrationServerName/
    applications/application_name/libraries/library_name/messageflows/message_flow_name
    ```

  - For publications on an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/integrationNodeName/Statistics/JSON/recordType/integrationServerName/
    applications/application_name/libraries/library_name/messageflows/message_flow_name
    ```

  where *integrationNodeName*, *integrationServerName*, *application_name*, *library_name*, and *message_flow_name* represent the integration node, integration server, application, library, and message flow on which you want to receive data. The *recordType* is the type of data collection on which you want to receive publications (SnapShot, Archive, or the topic wildcard "+" to receive both). The value that you specify for record type is case-sensitive, so if you choose to receive snapshot data, set the record type to `SnapShot`.

6. Decide the type of data collection that you want:
   - Snapshot
   - Archive
   - Snapshot and archive
7. Run the **mqsichangeflowstats** command to activate or deactivate the collection of snapshot data, archive data, or both.

   For example, to turn on snapshot statistics for message flow `flow2` in application `app1`, on the specified integration server, use the following command:

   ```
   mqsichangeflowstats -c active -s -i localhost -p 7600 -k app1 -f flow2
   ```

   To turn off archive and snapshot statistics for all message flows in all applications and libraries on the specified integration server, use the following command:

   ```
   mqsichangeflowstats -c inactive -a -s -i localhost -p 7600 -k "*" -y "*" -f "*"
   ```

   By default, the settings that you specify with this command persist when the integration server or node is restarted and when the message flow is redeployed. If you do not want the settings to persist, you can specify the **--non-persist** parameter. The **--non-persist** parameter causes the settings to remain in effect only until a restart or redeploy, at which point the settings that are specified in the `server.conf.yaml` or `node.conf.yaml` configuration file take effect.

   For more information, see command.

Results

When the command completes successfully, data collection for the specified resources is started:

- If you requested snapshot data, information is collected for approximately 20 seconds, and the results are written to the output specified in the `node.conf.yaml` or `server.conf.yaml` file.
- If you requested archive data, information is collected for the interval that is defined for the integration node or server (in the `node.conf.yaml` or `server.conf.yaml` file). The results are written to the specified output. The interval is reset and data collection starts again.

What to do next

You can now view the message flow statistics and accounting data that is collected, as described in “Viewing message flow statistics and accounting data” on page 2738.

- “Setting message flow accounting and statistics accounting origin” on page 2735
- “Viewing message flow statistics and accounting data” on page 2738

Setting message flow accounting and statistics accounting origin

You can request accounting origin support for the collection of message flow accounting and statistics data, by setting properties in the configuration file for your integration node (`node.conf.yaml`) or integration server (`server.conf.yaml`). You must also configure your message flows to provide the correct identification values that indicate what the data is associated with.

Before you begin

- Ensure that you have created a message flow; for more information, see “Creating a message flow” on page 574.
- Read the concept topic, “Message flow accounting and statistics accounting origin” on page 2702

About this task

Accounting and statistics data is associated with an accounting origin. For more information, see “Message flow statistics and accounting data” on page 2699 and “Message flow accounting and statistics accounting origin” on page 2702.

You can set a different value for every message flow for which data collection is active, or the same value for a group of message flows (for example, those in a single integration server, or associated with a particular client, department, or application suite).

The accounting origin setting is not used until you deploy the message flow or flows to the integration nodes on which they are to run. You can activate data collection, or modify it to request accounting origin support, before or after you deploy the message flow. You do not have to stop collecting data when you deploy a message flow that changes accounting origin.

To configure a message flow to specify a particular accounting origin, complete the following steps.

Procedure

1. Open the message flow with which you want to work.
2. Click **Selection** above the palette of nodes.
3. Right-click a Compute, Database, or Filter node in the editor view, then click **Open ESQL**.

   The associated ESQL file is opened in the editor view, and your cursor is positioned at the start of the correct module. You can include the required ESQL in any of these nodes, so decide which node in each message flow is the most appropriate for this action.

   To take advantage of the accounting origin support, include one of these nodes in each message flow for which you want a specific origin set. If you have not configured one of these three nodes in the message flow, you must add one at a suitable point (for example, immediately following the input node) and connect it to other nodes in the flow.
4. Update the ESQL in the node module to set an accounting origin.

   The integration node uses the origin identifier that is set in the Environment tree. You must set a value in the field with correlation name `Environment.Broker.Accounting.Origin`. This field is not created automatically in the Environment tree when the message is first received in the integration

node. The field is created only when you set it in an ESQL module that is associated with a node in the message flow.

If you do not set a value in the message flow, the default value `Anonymous` is used for all output. If you set a value in more than one place in the message flow, the value that you set immediately before the message flow terminates is used in the output data.

The code that you must add has the following form:

```
SET Environment.Broker.Accounting.Origin = "value";
```

You can set the identifier to a fixed value (as shown previously), or you can determine its value based on a dynamic value that is known only at run time. The value must be character data, and can be a maximum of 32 bytes. For example, you might set its value to the contents of a particular field in the message that is being processed (if you are coding ESQL for a Compute node, you must use correlation name InputBody in place of Body in the following example):

```
IF Body.DepartmentName <> NULL THEN
   SET Environment.Broker.Accounting.Origin = Body.DepartmentName;
END IF;
```

5. Save the ESQL module, and check that you have not introduced any errors.
6. Save the message flow, and check again for errors.
7. Now that `Environment.Broker.Accounting.Origin` is set, you must enable the Accounting Origin collection function by setting the **`accountingOrigin`** property in the .yaml configuration file for your integration node or integration server.
   a) Open the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or integration server by using a YAML editor.

      If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are not valid characters in YAML files and would cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   b) In the Statistics section of the `.yaml` configuration file, set the **`accountingOrigin`** property to `basic`.

   c) Restart the integration server for the changes to take effect.

      For information about how to start an integration server, see “Starting an integration server” on page 250.

Results

You are now ready to deploy the updated message flow; for more information, see “Deployment rules and guidelines” on page 2465. Accounting and statistics data records that are collected after the message flow has been deployed includes the origin identifier that you have set.

Viewing message flow accounting and statistics data collection parameters by using the `mqsireportflowstats` command

You can view the parameters that are currently in effect for message flow accounting and statistics data collection by using the **`mqsireportflowstats`** command.

Before you begin

- Read the concept topic about message flow accounting and statistics data.
- Ensure that message flow statistics and accounting data are being collected. For more information, see “Configuring the collection of message flow statistics by using the mqsichangeflowstats command” on page 2732.

Procedure

Issue the **mqsireportflowstats** command to view the parameters that are currently being used to control archive or snapshot data collection.

You can view the parameters for an integration node, an integration server, or an individual message flow.

For example:

```
mqsireportflowstats --admin-host localhost --admin-port 7600
--snapshot --verbose --all-applications --all-flows
```

Refer to the command for further examples.

Results

The command displays the current status.

What to do next

You can now modify the data collection parameters.

Modifying message flow accounting and statistics data collection settings by using the mqsichangeflowstats command

You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an integration server for which you are already collecting data. You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Before you begin

- Start collecting message flow statistics; see “Configuring the collection of message flow statistics by using the mqsichangeflowstats command” on page 2732
- Read the concept topic about “Message flow statistics and accounting data” on page 2699

About this task

To modify message flow accounting and statistics parameters:

Procedure

1. Decide which data collection parameters you want to change.

   You can modify the parameters that are in force for an integration node, an integration server, or an individual message flow.

2. Issue the **mqsichangeflowstats** command with the appropriate parameters to modify the parameters that are currently being used to control archive data collection or snapshot data collection.

   For example, to modify parameters to extend snapshot data collection to a new message flow MFlow2 in integration server IS2 for integration node INodeA, enter:

   ```
   mqsichangeflowstats INodeA -s -e IS2 -f MFlow2 -c active
   ```

   ```
   /F BrokerA,cs s=yes,e=EG2,f=MFlow2,c=active
   ```

   If you want to specify an accounting origin for archive data for a particular message flow in an integration server, enter:

   ```
   mqsichangeflowstats INodeA -a -e IS44 -f MFlowX -b basic
   ```

   ```
   /F BrokerA,cs a=yes,e=EG4,f=MFlowX,b=basic
   ```

   Refer to the command for further information.

Results

When the command completes successfully, the new parameters that you have specified for data collection are in effect. These parameters remain in effect until you stop data collection or make further modifications.

Viewing message flow statistics and accounting data

You can use the web user interface to view snapshot statistics and accounting data for independent integration servers and managed integration servers (which are managed by an integration node). You can also report statistics and accounting data for independent integration servers and managed integration servers to the user trace log, IBM MQ, and MQTT.

Before you begin

- Read the topic “Message flow statistics and accounting data” on page 2699.
- Start message flow statistics collection, as described in “Configuring the collection of message flow statistics by using a .yaml configuration file” on page 2729 and “Configuring the collection of message flow statistics by using the mqsichangeflowstats command” on page 2732.
- For information about how to subscribe to message flow statistics and accounting data, read “Subscribing to event message topics” on page 2813.

About this task

You can configure the statistics data to be published in a number of formats. For example, JSON, XML, CSV (for use in spreadsheets), and the user trace log.

Use the snapshot statistics and accounting data to monitor the performance of your integration servers at the message flow or message flow node level. You can view the statistics data for independent integration servers and managed integration servers in the App Connect Enterprise web user interface. The snapshot data that is displayed in the web user interface is updated automatically every 20 seconds. You can also report the data for independent integration servers or managed integration servers to the user trace log, IBM MQ, and MQTT.

For more information about how to start the web user interface for an integration server, see “Accessing the web user interface” on page 89.

You can display high-level statistical data that provides an overview of all the message flows in an integration server, or you can display statistical information that relates to a specific message flow. The level of information that is displayed depends on the type of resource that you select.

You can use the statistics data in these views to compare the performance of message flows in your applications. You can also use it to help you identify the possible reasons for reduced performance in a message flow.

Analyzing resource performance

You can collect statistics to assess the performance of resources used by integration servers.

Before you begin

Read the concept topic about resource statistics.

About this task

- “Managing resource statistics collection” on page 2741
- “Viewing resource statistics data” on page 2745

Resource statistics

Resource statistics are collected to record performance and operating details of resources that are used by integration servers.

As a system administrator, you can use the resource statistics to ensure that your systems are using the available resources in the most efficient manner. By monitoring systems and analyzing statistical trends, you can keep system resource usage within boundaries that you consider acceptable. You can also help to preempt situations where system resources are overstretched and might become unavailable. Analysis of the data that is returned potentially requires specialist skills and knowledge of each resource type.

If you detect that system resources are under pressure, you can examine the statistics to assess whether the cause of the concern is the use of those resources by processes in IBM App Connect Enterprise. You can collect data on one or more selected integration servers, or for all integration servers. When statistics collection is enabled, you might experience a minor degradation in operating performance of the integration servers for which you are collecting data.

When an integration node or server is created, the publication of resource statistics is enabled by default. The **reportingOn** property in the `node.conf.yaml` or `server.conf.yaml` file is explicitly set to `true`, which enables the publication of snapshot statistics to the web user interface. For integration nodes and servers that were created prior to IBM App Connect Enterprise Version 11.0.0.8, the publication of resource statistics was turned off by default. To enable the publication of statistics for your integration nodes or servers that were created prior to V11.0.0.8, edit the relevant `.conf.yaml` file and set the **reportingOn** property to `true`. For more information, see “Managing resource statistics collection” on page 2741.

Before resource statistics can be collected, you must ensure that the publication of events is enabled. If you are subscribing to resource statistics data, you must also ensure that a pub/sub broker is configured. For more information, see “Configuring the publication of event messages” on page 2807 and “Configuring the built-in MQTT pub/sub broker” on page 2811.

To start, stop, or check the status of resource statistics collection, you can also use the command and the command.

To set and check the properties that control where resource statistics are reported, use the command and the command.

You can view the resource statistics data for your integration nodes and servers by using the web user interface.

You can also specify that resource statistics data for the resource managers in your integration servers is written to files. For more information, see “Reporting resource statistics to files” on page 2744.

View statistics data by using the publish/subscribe method

You can view the output that is generated by statistics collection by using an application that subscribes to a publication message, which is published by the integration node every 20 seconds. The message contains the data that is collected for each integration server for which you activate statistics collection. The published message is available in XML format and in JSON format.

The topic for each message has the following structure:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/integrationNodeName/ResourceStatistics/integration_server_name
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/integrationNodeName/ResourceStatistics/integration_server_name
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/integrationNodeName/Statistics/JSON/Resource/integration_server_name
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/integrationNodeName/Statistics/JSON/Resource/integration_server_name
    ```

You can set up subscriptions for a specific integration server on a specific integration node. See the following examples:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/INODE/ResourceStatistics/default
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/INODE/ResourceStatistics/default
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/INODE/Statistics/JSON/Resource/default
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/INODE/Statistics/JSON/Resource/default
    ```

You can also use wildcards in the subscriptions to broaden the scope of what is returned. For example, to subscribe to reports for all integration servers on all integration nodes, use the following values:

- For XML format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/+/ResourceStatistics/#
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/+/ResourceStatistics/#
    ```

- For JSON format:
  - For an IBM MQ pub/sub broker:

    ```
    $SYS/Broker/+/Statistics/JSON/Resource/#
    ```

  - For an MQTT pub/sub broker:

    ```
    IBM/IntegrationBus/+/Statistics/JSON/Resource/#
    ```

For more information about subscribing to an integration server, see "Subscribing to event message topics" on page 2813.

For more information about the statistics that are reported for each resource manager, and the publication content, see "Resource statistics data" on page 2746.

Managing resource statistics collection

Use resource statistics data to monitor the performance and resource usage of your integration servers.

Before you begin

- Configure an integration server, by following the instructions described in "Configuring an integration server by modifying the server.conf.yaml file" on page 172.
- Create your message flow, package it into a BAR file, and then deploy it to your integration server. For more information, see "Creating a message flow" on page 574 and "Deploying integration solutions to a production environment" on page 2480.
- Read the concept topic, "Resource statistics" on page 2739.

About this task

You can activate the collection of resource statistics by modifying a `node.conf.yaml` or `server.conf.yaml` configuration file, by using the IBM App Connect Enterprise web user interface, or by running the **`mqsichangeresourcestats`** command.

You can view the resource statistics data in the web user interface. Alternatively, you can configure your integration servers to write resource statistics to files, as described in "Reporting resource statistics to files" on page 2744.

You can manage the collection of resource statistics data for active integration servers by using one of the following methods:

- "Managing resource statistics collection by modifying a server.conf.yaml file" on page 2741
- "Managing resource statistics collection by using the web user interface" on page 2742
- "Managing resource statistics collection by using a command" on page 2743

Managing resource statistics collection by modifying a server.conf.yaml file

About this task

You can start collecting resource statistics data for resources that are used by any active message flows that are deployed in an integration server. To start collecting the data, set the resource statistics properties in the `server.conf.yaml` file for the integration server. Alternatively, you can configure the collection of resource statistics for the integration node by modifying the properties in the `node.conf.yaml` file.

When you configure the resource statistics settings for your integration server, you can start and stop the collection of resource statistics dynamically, without restarting the integration server, by using the command. You can also see the current options for gathering resource statistics by using the command.

Procedure

Complete the following steps to configure the collection of resource statistics:

1. Open the `server.conf.yaml` configuration file for your integration server, by using a YAML editor.

   If you cannot access a YAML editor, you can edit the file by using a plain text editor. Ensure that you do not include any tab characters, which are not valid in YAML and would cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file. The properties that you need to set are in the **Statistics** section of the `.yaml` configuration file:

   ```
   Statistics:
        ...
      Resource:
      reportingOn: true         # Choose 1 of : true|false. Explicitly set to 'true' by
    default. If unset, defaults to 'false'.
      #outputFormat: ''         # Choose 'csvFile' or 'file' (for IIB v10 compatibility). If
    unset, defaults to ''.
   ```

2. Uncomment the **reportingOn** property and set it to `true`. This setting takes effect when the integration server is restarted. However, it can be changed dynamically (without restarting the integration server) by using the command.

   When a node`.conf.yaml` or `server.conf.yaml` file is created, the **reportingOn** property is explicitly set to `true`, and resource statistics for the integration node or server are published to the web user interface. If the **reportingOn** property is unset (with a value of `''`), the publication of resource statistics is turned off.

   In node`.conf.yaml` and `server.conf.yaml` files that were created prior to IBM App Connect Enterprise V11.0.0.8, the **reportingOn** property was set to `false` by default, so the publication of resource statistics was turned off. To enable the publication of resource statistics for those integration nodes or servers, edit the relevant `.conf.yaml` file and set the **reportingOn** property to `true`.

3. Optional: If you want the resource statistics to be published to a file (in addition to the web user interface), set the **outputFormat** property to `csvFile`. For more information, about publishing resource statistics to a file, see “Reporting resource statistics to files” on page 2744.

4. Restart the integration server for the changes to take effect.

   For more information about how to start an integration server, see “Starting an integration server” on page 250.

Managing resource statistics collection by using the web user interface

Procedure

1. Start the web user interface for your integration node; see “Accessing the web user interface” on page 89.

   The integration servers that are owned by the integration node are displayed as tiles under the **Servers** tab.

2. Click the **Open List of Options** icon for the required integration server.

   - If you are viewing all the available servers under the **Servers** tab, the **Open List of Options** icon appears within the tile for each integration server.

3. Click **Open** in the drop-down menu to view the content of the integration server under the **Contents** tab.

4. Click the **Open List of Options** icon.

   - If you are viewing the content of an integration server under the **Contents** tab, the **Open List of Options** icon appears in the title bar for the integration server.

5. Click the required option in the drop-down menu.
   - To start collecting resource statistics for this integration server, click **Resource statistics on** in the drop-down menu.
   - To stop collecting resource statistics for this integration server, click **Resource statistics off** in the drop-down menu.

   If resource statistics collection is turned on for the integration server, data is displayed in the **Resource Statistics** tab.

Managing resource statistics collection by using a command

Procedure

1. If your integration server is running on Linux, UNIX, or Windows systems, set up the correct command environment.

   For more information about how to complete this task, see "Setting up a command environment" on page 64.
2. Run one of the following commands:
   - To start or stop collecting resource statistics for a specific integration server, or for all integration servers, run the **mqsichangeresourcestats** command with the appropriate parameters.

     For example, to start collecting resource statistics for the default integration server for an integration node that is named INODE, enter the following command:

     ```
     mqsichangeresourcestats INODE -c active -e default
     ```

     For example, to stop collecting resource statistics for all integration servers on an integration node that is named INODE, enter the following command:

     ```
     mqsichangeresourcestats INODE -c inactive
     ```

     By default, the settings that you specify with this command persist when the integration server or node is restarted and when the message flow is redeployed. If you do not want the settings to persist, you can specify the **--non-persist** parameter, which causes the settings to remain in effect only until a restart or redeploy. After a restart or redeploy, the settings that are specified in the `server.conf.yaml` or `node.conf.yaml` configuration file take effect.

     For more information, see command.
   - To check the status of resource collection, run the **mqsireportresourcestats** command with the appropriate parameters.

     For example, to view status for the default integration server on INODE, enter:

     ```
     mqsireportresourcestats INODE -e default
     ```

     For further examples, see the command.
   - To set and check the properties that control where resource statistics are reported, use the **mqsichangeproperties** and **mqsireportproperties** commands.

What to do next

You can view the resource statistics in the web user interface, as described in "Viewing resource statistics data" on page 2745.

You can write a program that subscribes to a publication (single XML message) that returns the resource statistics data. For an example of the publication message, see Example XML output.

Reporting resource statistics to files

You can configure an integration server or integration node to send resource statistics directly to files in the file system.

About this task

You can configure an integration server to send resource statistics directly to files in the file system, by modifying the `server.conf.yaml` configuration file for the integration server and setting the resource statistics properties. You can configure the collection of resource statistics for the integration node by modifying the properties in the `node.conf.yaml` file.

If you enable this option, each resource manager writes a separate file that contains its resource statistics. Limits apply to the size of each file and if the limit for a file is reached, a new file is created. Limits also apply to the number of files that can be written for each resource manager, and if the limit for the number of files for the resource manager is reached, the resource manager then writes to a previous file in a circular fashion.

When you have configured the resource statistics settings for your integration server or integration node, you can start and stop the collection of resource statistics dynamically, without restarting the integration server, by using the command. You can also see the current statistics gathering options for resources by using the command.

For information about publishing resource statistics to an MQ queue manager or MQTT pub/sub broker, see “Configuring the publication of event messages” on page 2807.

Procedure

Configure an integration server to write resource statistics to files, by completing the following steps:

1. Open the configuration file for your integration server (`server.conf.yaml`) by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are not valid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   The properties that you need to set are in the **`Statistics`** section of the `.yaml` configuration file:

   ```
   Statistics:
      # Application message flows will by default inherit Snapshot and Archive values set here
        ...
        Resource:
           reportingOn: true            # choose 1 of : true|false. Explicitly set to 'true' by
    default. If unset, defaults to 'false'.
           outputFormat: 'csvFile'      # choose 'csvFile' or 'file' (for IIB v10
    compatibility). If unset, defaults to ''.
   ```

2. Set the following properties:

   a) Set the **`reportingOn`** property to `true`.

   b) Set the **`outputFormat`** property to either `csvFile` or `file`:

   csvFile

   The CSV data is organized in rows with columns for reported metrics. The first column contains the resource name and the remaining columns contain the reported properties for the named

resource. For resource managers that produce sub-resource data, extra rows are written for the summary and for each sub-resource. For example:

- The **Parsers** resource manager emits a row for the `Summary` and for each *message flow*.*parser* sub-resource.
- The **JVM** resource manager emits a row for the `Summary` and rows for the `Heap Memory`, `Non-Heap Memory`, `Garbage Collection - scavenge`, and `Garbage Collection - global` sub-resources.

file

The `file` format is supported for compatibility with IBM Integration Bus Version 10. Each resource manager emits a single row for each reporting interval.

For resource managers that produce sub-resource data, additional columns are added. The column titles are formed by prefixing the metric name with the resource name. For example, the **Parsers** resource manager emits columns `Summary_`*`metric_name`*, and for each sub-resource another set of columns is emitted in the form *message flow*.*parser*`_`*metric_name*.

Only the first row in each resource manager file contains the column titles, so titles are shown only for the summary and sub-resources that exist when the new file is written. Additional columns with extra sub-resources can appear in the file without being written in the first title row. In App Connect Enterprise Version 11, components are not assigned UUIDs, so these columns are populated with zeros.

The integration server writes the data to files at the following location:

```
serverWorkDirectory/config/common/resourceStats/
ResourceStats_integration_server_serverName_resource.txt.n
```

where:

- *serverWorkDirectory* is the value of the **--work-dir** parameter.
- *serverName* is the value of the **--name** parameter.
- *resource* is the name of the reported resource, such as JVM or JDBCConnectionPools.

3. Restart the integration server for the changes to take effect.

   For information about how to start an integration server, see “Starting an integration server” on page 250.

Results

You can view the resource statistics by accessing the files that each resource manager has written. The files are in `.csv` format.

Use the links in “Resource statistics data” on page 2746 for information about the resource statistics data that is collected.

Viewing resource statistics data

You can view resource statistics for independent integration servers and managed integration servers (which are owned by an integration node) in the IBM App Connect Enterprise web user interface. If you specify that the statistics are to be written to the file system, you can also view them in a file format.

Before you begin

- Read the concept topic, “Resource statistics” on page 2739.
- Start resource statistics collection, as described in “Managing resource statistics collection” on page 2741.

About this task

You can also view resource statistics collection by subscribing to the topic on which statistics are published. For more information, see “Subscribing to event message topics” on page 2813.

To view resource statistics in the web user interface, start a browser and load the user interface for your integration server. You can then browse the statistics that you are interested in. For more information about how to start the web user interface, see “Accessing the web user interface” on page 89.

Resource statistics data

Learn about the measurements for which data is returned when you activate resource statistics collection.

Statistics information is collected for the following resource types:

You can then view these statistics in the web user interface. You can also specify that resource statistics are written to files in the file system, as described in “Reporting resource statistics to files” on page 2744

Example of an XML publication for resource statistics

This example message shows an XML publication that contains resource statistics data.

A single XML publication includes all the statistics information that is reported for the resource types listed in “Resource statistics data” on page 2746. The publication message starts with information that identifies the integration server to which the statistics apply.

For information about subscribing to these publication messages, see “Subscribing to event message topics” on page 2813.

The following message is an example that shows three sections, one for each of three different resource types, and the values that are returned for each type. The output lines in this example have been split to improve readability.

```
<ResourceStatistics
  brokerLabel="STRESS1"
  brokerUUID="1a09649b-c9b9-4efe-a9c0-5275f6dd6c3f"
  executionGroupName="eg.EAS.JVM.1"
  executionGroupUUID="d6a3b481-2401-0000-0080-c5acc915fa62"
  collectionStartDate="2009-10-22" collectionStartTime="11:42:17"
  startDate="2009-10-23" startTime="17:59:36.152"
  endDate="2009-10-23" endTime="17:59:56.154"
  timezone="Europe/London">
  <ResourceType name="JVM">
    <resourceIdentifier name="summary"
      InitialMemoryInMB="32" UsedMemoryInMB="30"
      CommittedMemoryInMB="52" MaxMemoryInMB="-1"
      CumulativeGCTimeInSeconds="0"
      CumulativeNumberOfGCCollections="32"/>
    <resourceIdentifier name="Heap Memory"
      InitialMemoryInMB="32" UsedMemoryInMB="14"
      CommittedMemoryInMB="32" MaxMemoryInMB="256"/>
    <resourceIdentifier name="Non-Heap Memory"
      InitialMemoryInMB="0" UsedMemoryInMB="16"
      CommittedMemoryInMB="20" MaxMemoryInMB="-1"/>
    <resourceIdentifier name="Garbage Collection - J9 GC"
      CumulativeGCTimeInSeconds="0"
      CumulativeNumberOfGCCollections="32"/>
  </ResourceType>
  <ResourceType name="ODM">
    <resourceIdentifier name="summary"
      SuccessfulExecutions="2"
      FailedExecutions="0"
      RulesMatched="6" />
    <resourceIdentifier name="{SISPolicyProject}:ODM_Files(/RULEAPP_PROJ_2L_MULTIRULES/1.0/
ODM_2L_MULTIRULES/1.0)"
      SuccessfulExecutions="1"
      FailedExecutions="0"
      RulesMatched="3" />
    <resourceIdentifier name="{SISPolicyProject}:ODM_Files(/ODM_2L_MULTIRULES_892/1.0/
ODM_2L_MULTIRULESRuleset/1.0)"
      SuccessfulExecutions="1"
      FailedExecutions="0"
      RulesMatched="3" />
  </ResourceType>
  <ResourceType name="JMS">
    <resourceIdentifier name="summary"
      NumberOfOpenJMSConnections="3"
      NumberOfClosedJMSConnections="4"
      NumberOfOpenJMSSessions="12"
      NumberOfClosedJMSSessions="9"
      NumberOfMessagesReceived="7"
      NumberOfMessagesSent="19"
      NumberOfMessagesBrowsed="4"
      NumberOfJMSConnectionFailures="0"/>
    <resourceIdentifier name="JMSTest_ASYNCQCF"
      NumberOfOpenJMSConnections="1"
      NumberOfClosedJMSConnections="0"
      NumberOfOpenJMSSessions="3"
      NumberOfClosedJMSSessions="0"
      NumberOfMessagesReceived="7"
      NumberOfMessagesSent="4"
      NumberOfMessagesBrowsed="4"
      NumberOfJMSConnectionFailures="0"/>
    <resourceIdentifier name="JMSTest1_ASYNCQCF"
      NumberOfOpenJMSConnections="1"
      NumberOfClosedJMSConnections="0"
      NumberOfOpenJMSSessions="1"
      NumberOfClosedJMSSessions="0"
      NumberOfMessagesReceived="0"
      NumberOfMessagesSent="15"
      NumberOfMessagesBrowsed="0"
      NumberOfJMSConnectionFailures="0"/>
  </ResourceType>
  <ResourceType name="Security">
    <resourceIdentifier name="summary"
      TotalCacheEntries="10"
      TotalOperations="9"
      TotalSuccessfulOperations="6"
      TotalOperationsServicedByCache="3"/>
```

```
    <resourceIdentifier name="LDAP"
      TotalOperations="3"
      TotalSuccessfulOperations="1"
      TotalOperationsServicedByCache="0"
      TotalCacheEntries="0"/>
    <resourceIdentifier name="WS-Trust v1.3 STS"
      TotalOperations="6"
      TotalSuccessfulOperations="5"
      TotalOperationsServicedByCache="3"
      TotalCacheEntries="0"/>
  </ResourceType>
  <ResourceType name="Sockets">
    <resourceIdentifier name="summary"
      TotalMessages="826"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="1715189"
      AverageBytesSentPerSecond="85758"
      TotalBytesReceived="116879"
      AverageBytesReceivedPerSecond="5843"
      AverageBytesSentPerMessage="2076"
      AverageBytesReceivedPerMessage="141"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="826"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="826"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
    <resourceIdentifier name="9.146.149.86.7900"
      TotalMessages="413"
      AverageMessagesPerMinute="1239"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="837977"
      AverageBytesSentPerSecond="41898"
      TotalBytesReceived="60298"
      AverageBytesReceivedPerSecond="3014"
      AverageBytesSentPerMessage="2029"
      AverageBytesReceivedPerMessage="146"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="413"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="413"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
    <resourceIdentifier name="localhost.7900"
      TotalMessages="413"
      AverageMessagesPerMinute="1239"
      TotalSocketsOpened="0"
      AverageSocketsOpenedPerMinute="0"
      TotalBytesSent="877212"
      AverageBytesSentPerSecond="43860"
      TotalBytesReceived="56581"
      AverageBytesReceivedPerSecond="2829"
      AverageBytesSentPerMessage="2124"
      AverageBytesReceivedPerMessage="137"
      SentMessageSize_0-1KB="0"
      SentMessageSize_1KB-10KB="413"
      SentMessageSize_10KB-100KB="0"
      SentMessageSize_100KB-1MB="0"
      SentMessageSize_1MB-10MB="0"
      SentMessageSize_Over10MB="0"
      ReceivedMessageSize_0-1KB="413"
      ReceivedMessageSize_1KB-10KB="0"
      ReceivedMessageSize_10KB-100KB="0"
      ReceivedMessageSize_100KB-1MB="0"
      ReceivedMessageSize_1MB-10MB="0"
      ReceivedMessageSize_Over10MB="0"/>
  </ResourceType>
  <ResourceType name="Message parsers">
```

```
    <resourceIdentifier name="summary"
      Threads="1"
      ApproxMemKB="3"
      MaxReadKB="1"
      MaxWrittenKB="0"
      Fields="100"
      Reads="1"
      FailedReads="0"
      Writes="0"
      FailedWrites="0"/>
    <resourceIdentifier name="Flow1.XMLNSC"
      Threads="1"
      ApproxMemKB="3"
      MaxReadKB="1"
      MaxWrittenKB="0"
      Fields="100"
      Reads="1"
      FailedReads="0"
      Writes="0"
      FailedWrites="0"/>
  </ResourceType>
</ResourceStatistics>
```

All times are reported in local time, and the timezone used is reported by the timezone attribute in the standard TZ Area/Location format. The date is in the format yyyy-MM-dd, and time is in the format HH:mm:ss, independent of locale and timezone.

The collectionStartDate and collectionStartTime specify when resource statistics collection started, typically when the integration server was last started.

The endDate and endTime specify when the statistics data in the current message was gathered. The startDate and startTime is when the previous set of data was reported.

Resource statistics data: .NET Application Domains

Learn about the data that is returned for the .NET Application Domains resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

High-level memory usage is shown for each application domain that is running in the integration server. Statistics are updated only after a full blocking garbage collection.

Measurements	Description
name	The name of the application domain.
CurrentlyInUseMB	The memory used by the domain at time of the last garbage collection.
TotalAllocatedInMB	The total amount of memory allocated by the application domain since it started (including any memory that has been garbage-collected).
Id	The numeric ID for this application domain.
ApplicationBase	The directory currently being used to load assemblies for this domain.

Resource statistics data: .NET Garbage Collection

Learn about the data that is returned for the .NET Garbage Collection resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Gives detailed statistics on garbage collection, for the entire DotNet CLR. For details on interpreting these values, see the Microsoft documentation for .NET.

Measurements	Description
ExplicitGCCount	Indicates the number of garbage collections that were forced by external request.
Gen0CollectionsTaken	Indicates the number of garbage collections completed for generation 0.
Gen1CollectionsTaken	Indicates the number of garbage collections completed for generation 1.
Gen2CollectionsTaken	Indicates the number of garbage collections completed for generation 2.
CommittedInMB	The total number of Megabytes committed in all heaps.
ReservedInMB	The total number of Megabytes reserved in all heaps.
Gen0HeapSizeInMB	The size, in Megabytes, of the generation-zero heap.
Gen1HeapSizeInMB	The size, in Megabytes, of the generation-one heap.
Gen2HeapSizeInMB	The size, in Megabytes, of the generation-two heap.
LargeObjectHeapSizeInMB	The size, in Megabytes, of the large object heap.
PromotedFromGen0InMB	The size, in Megabytes, of the objects promoted from generation 0 to generation 1.
PromotedFromGen1InMB	The size, in Megabytes, of the objects promoted from generation 1 to generation 2.

Resource statistics data: CICS

Learn about the data that is returned for the CICS resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A CICSRequest node calls CICS Transaction Server for z/OS programs. Use these resource statistics to review how many successful and unsuccessful calls the node is making to CICS, how many unsuccessful requests are related to security issues such as authentication failures, and how often connection attempts fail.

A statistics summary is returned for the whole integration server, followed by a breakdown for each named CICS Transaction Server for z/OS region.

The following table describes the measurements that are returned for each CICS region. Each row of statistics represents a CICS region, which is identified either by a policy, or by the CICSRequest node `CICS server` URL.

- In this example, the policy is identified by *myCICSConnectionService*, and the integration node is identified by APPLID*BRKApp* and qualifier *BRKQual*:

```
myCICSConnectionService.BRKApp.BRKQual
```

- In this example, the CICSRequest node `CICS server` URL is identified by *tcp://mycicsregion.com:12345*:

```
tcp://mycicsregion.com:12345
```

Measurements	Description
RequestSuccess	The number of requests to CICS that are successful.

Measurements	Description
RequestFailures	The total number of requests to CICS that fail. The value that is returned by this measurement does not include connection failures. Connection failure values are returned by the `ConnectionAttemptFailures` measurement.
RequestSecurityFailures	The number of failed requests to CICS that were caused by security issues, such as authentication failures.
ConnectionAttemptFailures	The number of connection attempts that fail.

Resource statistics data: Connect Direct

Learn about the data that is returned for the Connect Direct resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are available for all Connect:Direct server activity, together with one line for each policy that is used.

The following table describes the measurements that are returned.

Measurements	Description
connectionDetails	The host name and port for the Connect:Direct server that is being connected to, in the format `<hostname>:port`. For the summary line, this property is set to an empty string.
inboundTransfers	The number of transfers received by the Connect:Direct server selected for processing in a message flow.
inboundBytes	Total number of bytes received in transfers, selected from Connect:Direct server.
nohitTransfers	The number of transfers received by the Connect:Direct server not selected for processing in a message flow. Reasons for not processing a transfer include: • The IBM App Connect Enterprise system has no access to the transferred file. • A deployed CDInput node has a filter that matches the transferred file. This number gives the total for the whole integration node and not just this integration server.
outboundTransfers	The number of transfers sent to the Connect:Direct server.
outboundBytes	Total number of bytes contained in transfers sent to the Connect:Direct server.

Resource statistics data: CORBA

Learn about the data that is returned for the CORBA resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A CORBARequest node calls a CORBA server. Use these resource statistics to review how many calls the node is making to the CORBA server, and how many of those calls are successful or result in CORBA exceptions. A statistics summary is returned for the whole integration server.

The following table describes the measurements that are returned for each CORBA node. The measurements apply to the number of calls since the integration server started.

Measurements	Description
OutboundInvocations	The total number of calls made to a CORBA server.
OutboundSuccessfulInvocations	The number of calls to the CORBA server that are successful.
OutboundCorbaExceptions	The number of calls to the CORBA server that result in CORBA exceptions.

Resource statistics data: File

Learn about the data that is returned for the file resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A summary is displayed for the local file system of any file actions done by any type of file node. It does not have any other entries; there is only one local file system. Remote file systems are reported in FTEAgent and FTP resource statistics.

Measurements	Description
FilesRead	The number of files successfully read by any file node
RecordsRead	The number of records read by either a file input node, FTE input node, or a file read node
BytesRead	The total number of bytes read by either a file input node, FTE input node, or a file read node
FilesCreated	The number of files successfully created by a file output node
RecordsWritten	The number of records written to files by a file output node or FTE output node
BytesWritten	The total number of bytes written by a file output node or FTE output node

Resource statistics data: FTE Agent

Learn about the data that is returned for the FTE Agent resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The FTE agent is an embedded FTE component that allows IBM App Connect Enterprise to send and receive files by using IBM MQ File Transfer Edition. A one-to-one mapping exists between an integration server and its FTE agent. Therefore, the resource manager statistics provide an accurate picture of the IBM MQ File Transfer Edition activity of that integration server.

The following table describes the measurements that are returned for the agent.

Measurements	Description
inboundTransfers	The number of transfers received by the agent.
outboundTransfers	The number of transfers sent by the agent.
inboundBytes	The number of bytes received by the agent.

Measurements	Description
outboundBytes	The number of bytes sent by the agent.

Resource statistics data: FTP

Learn about the data that is returned for the FTP resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A summary is displayed of file transfers that occur in the execution and a list for each server used.

Measurements	Description
Protocol	The protocol used: either FTP or SFTP
FTPGets	The number of transfers from a remote server to the file system of the integration node
BytesReceived	The number of bytes transferred from a remote server to the file system of the integration node
FTPPuts	The number of transfers from the file system of the integration node to a remote server
BytesSent	The number of bytes transferred from the file system of the integration node to a remote server **Note:** The FTPPuts counter can increase in two cases: either when a new file is transferred, or when an append is made to an existing file.

Resource statistics data: Global Cache

Learn about the data that is returned for the Global Cache resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The global cache resource manager contains data about the interactions between message flows in a named integration server and the cache. Statistics are available for all activity in the embedded global cache and external WebSphere eXtreme Scale grids. One line of statistics is shown for each policy that is used to connect to an external grid.

The following table describes the measurements that are returned.

Measurements	Description
ConnectionFailures	The number of failed attempts to connect from this integration server to the named cache.
Connects	The number of successful attempts that were made from this integration server to the named cache.
FailedActions	The number of failed map operations by message flows in this integration server on the named cache.
MapReads	The total number of read operations that were completed by message flows in this integration server on the named cache.

Measurements	Description
MapRemoves	The total number of remove operations that were completed by message flows in this integration server on the named cache. This count is incremented by removing or updating an entry.
MapsUsed	The total number of maps that are used by message flows in this integration server in the named cache. This count is incremented each time a different map name is used in this integration server.
MapWrites	The total number of write operations that were completed by message flows in this integration server on the named cache. This count is incremented by putting or updating an entry.
Name	The name of the WebSphere eXtreme Scale grid for this cache. For the embedded global cache, this name is "WMB". For an external grid, this name consists of the name of the policy that is used to connect to the grid, and the name of the grid itself. For example, if you use a policy called "remoteCacheServer" to connect to a grid called "Grid1", this name property has the value `remoteCacheServer(Grid1)`.
TotalMapActions	The total number of map operations that were completed by message flows in this integration server on the named cache. This number includes reads, writes, removes, and key checks.

Resource statistics data: Java Virtual Machine (JVM)

Learn about the data that is returned for the JVM resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Each integration server starts its own Java Virtual Machine (JVM), which provides support for all Java activities in that integration server. Some Java activity is present in the integration server even if you have not yet deployed or started message flows in that group. Use these resource statistics to review how much memory is in use by the JVM, and how often garbage collection might be occurring in the integration server.

Statistics are collected for the following JVM resources:

- Heap memory
- Non-heap memory
- Garbage collection

A summary is also provided, which adds the values in the three groups.

The special value `-1` is returned for measurements that are undefined or are not set.

The following table describes the measurements that are returned for each of the listed resources. The garbage collection measurements are cumulative, and continue to increase until you stop statistics collection.

Measurements	Resource	Description
InitialMemoryInMB	• Heap memory • Non-heap memory	The initial amount of memory that the JVM requests from the operating system for memory management during startup. Its value might be undefined.

Measurements	Resource	Description
UsedMemoryinMB	• Heap memory • Non-heap memory	The amount of memory that is currently in use.
CommittedMemoryInMB	• Heap memory • Non-heap memory	The amount of memory that is allocated to the JVM by the operating system.
MaxMemoryInMB	• Heap memory • Non-heap memory	The maximum amount of memory that can be used for memory management. Its value might be undefined.
CumulativeGCTimeInSeconds	Garbage collection	The accumulated garbage collection elapsed time in seconds for this instance of the JVM. Its value might be undefined.
CumulativeNumberOfGCCollections	Garbage collection	The total number of garbage collections that have occurred for this instance of the JVM. Its value might be undefined.

Resource statistics data: JDBC Connection Pools

Learn about the data that is returned for the JDBC Connection Pools resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

A statistics summary is returned for each JDBC provider defined in the integration node policy that is switched to use connection pools. The following examples are named JDBC data sources:

- DB2
- Oracle
- Microsoft SQLServer

The following table describes the measurements that are returned for each JDBC Providers policy.

Measurements	Description
NameOfJDBCProvider	The name of the JDBC Providers policy that is using connection pooling. If there are no connection pools started, this defaults to none.
MaxSizeOfPool	The maximum size of the connection pool.
ActualSizeOfPool	A snapshot of the number of connections currently in the pool when the statistics were reported.
CumulativeRequests	A count of the number of requests received by the connection pool during this accounting period.
CumulativeDelayedRequests	The number of times a request for a connection could not be satisfied immediately, because the number of allocated connections reached the maximum pool size and no connections are currently available.
MaxDelayInMilliseconds	The maximum time a caller waited for a connection to be allocated, in milliseconds.
CumulativeTimedOutRequests	A count of the number of requests for connections that could not be satisfied within 15 seconds.

Resource statistics data: JMS

Learn about the data that is returned for the JMS resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are reported for each JMS connection factory and JNDI bindings location. XA connections are distinguished from non-XA connections for the same connection factory and JNDI bindings location. Each resource is named either *ConnectionFactoryName_JNDIBindingsLocation* or *ConnectionFactoryName_JNDIBindingsLocation* (XA).

The following table describes the measurements that are returned for each of the listed resources.

Measurements	Description
NumberOfOpenJMSConnections	The current number of open JMS connections.
NumberOfClosedJMSConnections	The total number of JMS connections that were closed since the last integration server restart.
NumberOfOpenJMSSessions	The current number of open JMS sessions.
NumberOfClosedJMSSessions	The total number of JMS sessions that were closed since the last integration server restart.
NumberOfMessagesReceived	The total number of messages received by JMSInput or JMSReceive nodes.
NumberOfMessagesSent	The total number of messages sent by JMSOutput nodes.
NumberOfMessagesBrowsed	The total number of messages browsed by JMSReceive nodes.
NumberOfJMSConnectionFailures	The total number of attempted JMS connections that failed since the last integration server restarts.

Resource statistics data: MQTT

Learn about the data that is returned for the MQTT resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are reported for each MQTT server to which a connection exists.

The following table describes the measurements that are returned.

Measurements	Description
OpenConnections	The current number of open connections to an MQTT server.
ClosedConnections	The total number of connections to an MQTT server that were closed since the last integration server restart.
MessagesReceived	The total number of MQTT messages received by MQTTSubscribe nodes.
MessagesSent	The total number of MQTT messages sent by MQTTPublish nodes.
BytesReceived	The total amount of data received by MQTTSubscribe nodes.
BytesSent	The total amount of data sent by MQTTPublish nodes.

Measurements	Description
FailedConnections	The total number of attempted connections to an MQTT server that have failed since the last integration server restart.

Resource statistics data: ODBC

Learn about the data that is returned for the ODBC resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are reported for each ODBC DSN that was accessed since the integration server started. XA connections are distinguished from non-XA connections from the same DSN. Each resource is named either *DSN* or *DSN* (XA).

Measurements	Description
ExecuteSuccess	The total number of times any statement was run against this DSN.
ExecuteFailure	The total number of times any statement failed against this DSN.
ActiveConnections	The number of connections currently open to this DSN.
ClosedConnections	The number of connections to this DSN that were ever open, but are now closed. This figure includes connections closed due to an error, forced closed by the DBMS or closed by integration node because it was no longer required (for example, thread idle for 60 seconds).
ConnectionErrors	The number of times a connection to this DSN was detected to have a connection error (which would have caused the error to be closed and therefore also contributed to the closed connections measurement).

Resource statistics data: ODM

Learn about the data that is returned for the ODM resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Statistics are reported as an overall summary and for each unique ODM ruleset since the integration server started. A sub-resource statistics record is issued for each ruleset, in the form *ODM Policy*(*ruleset path*). For example, {PolicyProject}:ODM(/RuleSet1/1.0/Rule1/1.0).

Measurements	Description
SuccessfulExecutions	The number of successful executions of a ruleset.
FailedExecutions	The number of failed executions of a ruleset.
RulesMatched	The total number of rules matched during all executions of a ruleset.

For more information about using resource statistics, see “Resource statistics” on page 2739, and for information about using ODM rules with IBM App Connect Enterprise, see “Using Operational Decision Manager (ODM) business rules” on page 1241.

Resource statistics data: Parsers

Learn about the data that is returned for the parsers resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

All message flows in an integration server create parsers to parse and write input and output messages. Use the Parsers statistics to see how much resource is being used by the message trees and bit streams that these parsers own.

A statistics summary is returned, followed by an entry for accumulation by parser type for each message flow. The rows are named in the style *<Message Flow>.<Parser>*. A row is shown for every parser type used by that message flow. Additional instances are included in the accumulated statistics for each message flow.

The following table describes the statistics that are returned for each message flow parser since the integration server was last restarted.

Measurements	**Description**
Threads	The number of message flow threads that contributed to the statistics for a message flows parser type accumulation.
ApproxMemKB	The approximate amount of user data-related memory used for the named message flow parser type. It is not possible to calculate the exact amount of memory used by a parser.
MaxReadKB	Shows the largest bit stream parsed by the parser type for the named message flow.
MaxWrittenKB	Shows the largest bit stream written by the parser type for the named message flow.
Fields	Shows the number of message fields associated with the named message flow parser type. These fields are retained by the parser and are used for constructing the message trees.
Reads	The number of successful parses that were completed by the named message flow parser type.
FailedReads	The number of failed parses that occurred for the named message flow parser type.
Writes	The number of successful writes that were completed by the named message flow parser type.
FailedWrites	The number of failed writes that occurred for the named message flow parser type.

Resource statistics data: Security

Learn about the data that is returned for the Security resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

When a message flow is configured with a security profile, requests are typically made to a security provider or Security Token Service (STS) to process and approve authentication, mapping, or authorization. Use the security resource statistics to review the number of requests that are made, how many of those requests are successful, and how many are being serviced from the security cache.

A statistics summary is returned for the whole integration server, followed by a breakdown for each named security provider. Examples of named security providers are WS-Trust v1.3 STS and LDAP.

The following table describes the measurements that are returned for each provider.

Measurements	Description
TotalOperations	The number of security operations (authentication, mapping, or authorization) since collection started. A security profile with both authentication and authorization counts as two operations.
TotalSuccessfulOperations	The number of security operations (authentication, mapping, or authorization) that were approved.
TotalOperationsServicedByCache	The number of security operations (authentication, mapping, or authorization) that were serviced from the security cache (without accessing the STS directly).
TotalCacheEntries	The total number of security operation result entries in the security cache. A security operation is defined in the security profile as authentication, mapping, or authorization. A cache entry might include a returned security token.

Resource statistics data: SOAP Input

Learn about the data that is returned for the SOAP Input resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

The SOAPInput and SOAPReply nodes send and receive SOAP messages. Use the SOAPInput resource statistics to review how many inbound messages the SOAPInput node is receiving, how many replies the SOAPReply node is sending, and how many of those calls are successful or result in SOAP Faults, on a per-operation basis. Statistics for the SOAP nodes are collected with both HTTP and JMS transport. You can review the name of the applied policy set if one is defined. A statistics summary is returned for the whole integration server.

The following table describes the measurements that are returned for the SOAPInput and SOAPReply nodes. These statistics do not include messages which timed out.

Measurements	Description
Name	This measurement takes one of the following values: • "Summary" • *Flow*.*Node*.*Operation* • [Undeployed].*Flow*.*Node*.*Operation* where *Flow* is the name of your message flow, *Node* is the node name, and *Operation* is the name of the operation.
InboundMessagesTotal	The total number of SOAP messages received from the client. This measurement is equal to the sum of `InboundMessagesMadeFlow` and `InboundMessagesFaultedBeforeFlow`.
RepliesSentTotal	The total number of SOAP replies sent back to the client. This measurement is equal to the sum of `SuccessfulRepliesSent` and `FaultRepliesSent`.
InboundMessagesMadeFlow	The number of messages that made the flow without faulting.

Measurements	Description
InboundMessagesFaultedBeforeFlow	The number of messages that faulted before reaching the flow. This measurement includes input messages that are sent down the Failure terminal.
SuccessfulRepliesSent	The number of successful replies, without SOAP Fault, sent to the client.
FaultRepliesSent	The number of SOAP Fault replies sent to client. These faults can be user-defined faults or integration node exceptions.
PolicySetApplied	The name of the policy set if one was defined.

Resource statistics data: Sockets

Learn about the data that is returned for the Sockets resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Outbound sockets are used by the integration server when a message is sent out through a SOAP, SCA, or HTTP request node. SOAP and SCA nodes always use keepalive sockets; therefore outbound sockets are reused for many requests. HTTPRequest nodes use keepalive sockets only if you set the property Enable HTTP/1.1 keep-alive. Use these resource statistics to review whether you are reusing outbound sockets, and to see the size and volume of message data that is flowing through those sockets.

A statistics summary is returned, followed by an entry for each outbound socket endpoint, which is defined by its URL. Examples of endpoints are `localhost:7080` and `www.soaphub.org:80`.

The following table describes the measurements that are returned for each endpoint that was accessed since the integration server was last restarted.

Measurements	Description
TotalSockets	The number of outbound sockets that have been opened since the last integration server restart.
TotalMessages	The number of requests for a socket; for example, from a SOAPRequest node.
TotalDataSent_KB	The number of bytes sent, in kilobytes (KB).
TotalDataReceived_KB	The number of bytes received, in kilobytes (KB).
SentMessageSize_0-1KB SentMessageSize_1KB-10KB SentMessageSize_10KB-100KB SentMessageSize_100KB-1MB SentMessageSize_1MB-10MB SentMessageSize_Over10MB	The number of messages sent in each size range. For example, a message of 999 bytes is counted in SentMessageSize_0-1KB; a message of 1000 bytes is counted in SentMessageSize_1KB-10KB.
ReceivedMessageSize_0-1KB ReceivedMessageSize_1KB-10KB ReceivedMessageSize_10KB-100KB ReceivedMessageSize_100KB-1MB ReceivedMessageSize_1MB-10MB ReceivedMessageSize_Over10MB	The number of messages received in each size range.

Resource statistics data: TCPIP Client Nodes

Learn about the data that is returned for the TCPIP Client Nodes resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Use these statistics to get a view of the level of activity and health of TCP/IP nodes in an integration server. You can see how many connection managers are active and reporting statistics, and the following details from each:

Measurements	Description
OpenConnections	The current number of open connections
ClosedConnections	The total number of connections that were closed since the last integration server restart
MessagesReceived	The total number of messages received (by TCPIPClientInput or TCPIPClientReceive nodes)
MessagesSent	The total number of messages sent (by TCPIPClientOutput nodes)
BytesSent	The total amount of data sent (by TCPIPClientOutput nodes), excluding SSL wrappers.
BytesReceived	The total amount of data received (by TCPIPClientInput or TCPIPClientReceive nodes), excluding SSL wrappers.
FailedConnections	The total number of attempted connections that failed since the last integration server restarts.

Resource statistics data: TCPIP Server Nodes

Learn about the data that is returned for the TCPIP Server Nodes resource type when you activate resource statistics collection.

You can view these statistics in the web user interface, or you can write a program that subscribes to a publication (single XML message) that returns this data. For an example of the publication message, see Example XML output.

Use these statistics to get a view of the level of activity and health of TCP/IP nodes in an integration server. You can see how many connection managers are active and reporting statistics, and the following details from each:

Measurements	Description
OpenConnections	The current number of open connections
ClosedConnections	The total number of connections that were closed since the last integration server restart
MessagesReceived	The total number of messages received (by TCPIPServerInput or TCPIPServerReceive nodes)
MessagesSent	The total number of messages sent (by TCPIPServerOutput nodes)
BytesSent	The total amount of data sent (by TCPIPServerOutput nodes), excluding SSL wrappers.
BytesReceived	The total amount of data received (by TCPIPServerInput or TCPIPServerReceive nodes), excluding SSL wrappers.

Measurements	Description
FailedSSLConnections	The total number of attempted inbound SSL connections from external clients that failed or were refused since the last integration server restart.

Tuning the integration node

You can complete several tasks that enable you to tune different aspects of the integration node performance.

Before you begin

Ensure that the following requirements are met:

- Your user ID has the correct authorizations to perform the task. Refer to Security requirements for administrative tasks.

About this task

Select the tasks that are relevant to your enterprise:

Procedure

- "Setting the JVM heap size" on page 2762
- "Increasing the stack size on Windows, Linux, and UNIX systems" on page 2762

Setting the JVM heap size

When you start an integration server, it creates a Java virtual machine (JVM) for executing a Java user-defined node.

Before you begin

- Check the resource statistics data to monitor resource usage and identify potential problems with performance; for more information, see "Resource statistics" on page 2739.

About this task

You can pass parameters to the JVM to set the minimum and maximum heap sizes; the default maximum heap size is 256 MB. To give more capacity to a message flow that is going to process large messages, reduce the minimum JVM heap size to allow the main memory heap to occupy more address space.

Increase the maximum heap size only if you use Java intensively with, for example, user-defined nodes.

Use caution when you set the maximum heap size, because the Java Runtime Environment takes the values for its initial, maximum, and current heap sizes to calculate how frequently it drives garbage collection. A large maximum heap size drives garbage collection less frequently. If garbage collection is driven less frequently, the heap size associated with the integration server continues to grow.

Use the information on JVM parameter values on the **mqsichangeproperties** command to set the heap size that you require.

Increasing the stack size on Windows, Linux, and UNIX systems

Increase the stack size on Windows, Linux, and UNIX systems by setting the **MQSI_THREAD_STACK_SIZE** environment variable to an appropriate value.

About this task

When you restart integration nodes that are running on the system, they use the new value.

The value, in bytes, of **MQSI_THREAD_STACK_SIZE** that you set is used for every thread that is created in a DataFlowEngine process. If the integration server has many message flows assigned to it, and you set a large value for **MQSI_THREAD_STACK_SIZE**, the DataFlowEngine process needs a large amount of storage for the stacks.

Set this environment variable to at least 48 KB. The default values are:

Linux and UNIX
1 MB

AIX
2 MB

z/OS
50 KB

Windows
Determined by the operating system.

Optimizing message flow throughput

Each message flow that you design must provide a complete set of processing for messages received from a particular source. This design might result in very complex message flows that include large numbers of nodes that can cause a performance overhead, and might create potential bottlenecks. You can increase the number of message flows that process your messages to provide the opportunity for parallel processing and therefore improved throughput.

About this task

The operation mode of your integration servers can affect the number of message flows that you can use; see features and “Restrictions that apply in each operation mode” on page 5 for more information.

You can also consider the way in which the actions taken by the message flow are committed, and the order in which messages are processed.

The web user interface enables you to view message flow statistics and accounting data, including the message flow throughput and the CPU and elapsed times for transactions in the flow. See “Viewing message flow statistics and accounting data” on page 2738 for more information about how you can access the statistical data.

Consider the following options for optimizing message flow throughput:

Multiple threads processing messages in a single message flow
When you deploy a message flow, the integration server automatically starts an instance of the message flow for each input node that it contains. This behavior is the default. If you have a message flow that handles a very large number of messages, you can enable more messages to be processed by allocating more instances of the flow.

You can update the `Additional Instances` property of the deployed message flow in the BAR file; the integration server starts additional copies of the message flow on separate threads, providing parallel processing. This option is the most efficient way of handling this situation if you are not concerned about the order in which messages are processed.

If the message flow receives messages from an IBM MQ queue, you can influence the order in which messages are processed by setting the `Order Mode` property of the MQInput node:

- If you set `Order Mode` to By User ID, the node ensures that messages from a specific user (identified by the UserIdentifier field in the MQMD) are processed in guaranteed order. A second message from one user is not processed by an instance of the message flow if a previous message from this user is currently being processed by another instance of the message flow.
- If you set `Order Mode` to By Queue Order, the node processes one message at a time to preserve the order in which the messages are read from the queue. Therefore, this node behaves as though you have set the `Additional Instances` property of the message flow to zero.

- If you set `Order Mode` to `User Defined`, you can order messages by any message element, by setting an XPath or ESQL expression in the Order field location property. The node ensures that messages with the same value in the order field message element are processed in guaranteed order. A second message with the same value in the order field message element is not processed by an instance of the message flow if a previous message with the same value is currently being processed by another instance of the message flow.

  If the field is missing, an exception is raised, and the message is rolled back. NULL and empty values are processed separately, in parallel.

  If you set `Order Mode` to By User ID or `User Defined`, and the message flow uses transformation nodes, it is advisable to set the `Parse Timing` to Immediate.

The scope of the message flow

You might find that, in some circumstances, you can split a single message flow into several different flows to reduce the scope of work that each message flow performs. If you do split your message flow, be aware that it is not possible to run the separate message flows in the same unit of work, and if transactional aspects to your message flow exist (for example, the updating of multiple databases), this option does not provide a suitable solution.

The following two examples show when it might be beneficial to split a message flow:

1. In a message flow that uses a RouteToLabel node, the input queue might increase in size, indicating that work is arriving faster than it is being processed; this might indicate a need for increased processing capacity. You can use another copy of the message flow in a second integration server, but this option is not appropriate if you want all of the messages to be handled in the order in which they are shown on the queue. You can consider splitting out each branch of the message flow that starts with a Label node by providing an input queue and input node for each branch. This option might be appropriate, because when the message is routed by the RouteToLabel node to the relevant Label node, it has some level of independence from all other messages.

   You might also need to provide another input queue and input node to complete any common processing that the Label node branches connect to when unique processing has been done.
2. If you have a message flow that processes very large messages that take a considerable time to process, you might be able to:

   a. Create other copies of the message flow that use a different input queue (you can set this option up in the message flow itself, or you can update this property when you deploy the message flow).

   b. Set up IBM MQ queue aliases to redirect messages from some applications to the alternative queue and message flow.

   You can also create a new message flow that replicates the function of the original message flow, but only processes large messages that are immediately passed on to it by the original message flow, that you modified to check the input message size and redirect the large messages.

The frequency of commits

If a message flow receives input messages on an IBM MQ queue, you can improve its throughput for some message flow scenarios by modifying its default properties after you have added it to a BAR file. (These options are not available if the input messages are received by other input nodes; commits in those message flows are performed for each message.)

The following properties control the frequency with which the message flow commits transactions:

- `commitCount`. This property represents the number of messages processed from the input queue per message flow thread, before an MQCMIT is issued.
- `commitInterval`. This property represents the time interval that elapses before an MQCMIT is started.

Optimizing message flow response times

You can use different solutions to improve message flow response times.

Before you begin

Read the following concept topics:

- "Analyzing message flow performance" on page 2699
- "Message flow nodes" on page 484

About this task

When you design a message flow, the flexibility and functional capabilities of the built-in nodes often mean that there are several ways to achieve the processing and results that you require. You might find that different solutions deliver different levels of performance and, if performance is an important consideration for you, take it into account when designing your message flow

Your applications can perceive performance in either of these ways:

- The response time indicates how quickly each message is processed by the message flow. The response time is particularly influenced by how you design your message flows. Response time is discussed in this topic.
- The throughput indicates how many messages of particular sizes can be processed by a message flow in a specified time. The throughput is mainly affected by configuration and system resource factors, and is discussed in "Optimizing message flow throughput" on page 2763, with other domain configuration information.

The web user interface enables you to view message flow statistics and accounting data, including the message flow throughput and the CPU and elapsed times for transactions in the flow. See "Viewing message flow statistics and accounting data" on page 2738 for more information about how you can access the statistical data.

Several aspects influence message flow response times. However, as you create and modify your message flow design to arrive at the best results for your specific business requirements, also consider the eventual complexity of the message flow. The most efficient message flows are not necessarily the easiest to understand and maintain; experiment with the solutions available to arrive at the best balance for your needs.

Several factors influence message flow response times:

The number of nodes that you include in the message flow

Every node increases the amount of processing required in the integration server, therefore, consider the content of the message flow carefully, including the use of subflows.

Use as few nodes as possible in a message flow; every node that you include in the message flow increases the amount of processing required in the integration server. The number of nodes in a single flow has an upper limit, which is governed by system resources, particularly the stack size.

How the message flow routes and processes messages

In some situations, you might find that the built-in nodes, and perhaps other nodes that are available in your system, provide more than one way of providing the same function. Choose the simplest configuration. Where a message flow is required to process more than a single type of record, you can create an easily extendible framework by developing a message flow structure in which there is a parse of the message to determine the type, followed by a RouteToLabel node and Label nodes for each of the types. Where a higher number of label nodes is expected, consider implementing the message parse and Label selection in one message flow, and the processing of each of the label types into separate message flows. The interface between these two flows would be through a queue.

If your message flow includes loops
Avoid loops of repeating nodes, which can be very inefficient and can cause performance and stack problems. You might find that a Compute node with multiple PROPAGATE statements avoids the need to loop around a series of nodes.

The efficiency of the ESQL
Check all the ESQL code that you have created for your message flow nodes. As you develop and test a node, you might maintain statements that are not required when you have finalized your message processing. You might also find that something you have coded as two statements can be coded as one. Taking the time to review and check your ESQL code might provide simplification and performance improvements.

The use of persistent and transactional messages
Persistent messages are saved to disk during message flow processing. You can avoid this situation by specifying that messages are non-persistent on input, output, or both. If your message flow is handling only non-persistent messages, check the configuration of the nodes and the message flow itself; if your messages are non-persistent, transactional support might be unnecessary. The default configuration of some nodes enforces transactionality; if you update these properties and redeploy the message flow, response times might improve.

Message size
A larger message takes longer to process. If you can split large messages into smaller units of information, you might be able to improve the speed at which they are handled by the message flow.

Message format
Although IBM App Connect Enterprise supports multiple message formats, and provides facilities that you can use to transform from one format to another, this transformation increases the amount of processing required in the integration server. Make sure that you do not perform any unnecessary conversions or transformations.

Troubleshooting performance problems

Follow this guidance to resolve common problems with IBM App Connect Enterprise performance.

About this task

- **Scenario:** You are experiencing problems with performance, such as:
  - Poor response times in the IBM App Connect Enterprise Toolkit when developing message flows
  - Poor response time at deployment
  - Individual messages taking a long time to process
  - Poor overall performance, or performance that does not scale well
- **Solution:** Possible solutions are:
  - Tune the integration node
  - Speed up IBM MQ persistent messaging by optimizing the I/O (input/output)
  - Speed up database access by optimizing I/O
  - Increase system memory
  - Use additional instances or multiple integration servers
  - Optimize ESQL statements for best performance

The reports in IBM MQ Family Category 2 (freeware) SupportPacs contain additional advice about performance, and they are available for download from the IBM MQ SupportPacs web page.

The following topics contain information about dealing with some common performance problems:

- "You are experiencing poor startup times for the IBM App Connect Enterprise Toolkit" on page 2769
- "You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large projects" on page 2769

- "Performance is reduced when you run web services with small message sizes" on page 2770
- "The PutTime reported by IBM MQ on z/OS, and other times or timestamps are inconsistent" on page 2770

The following topics contain guidance to help you optimize the performance of your code and message flows:

- "Message flow design and performance" on page 2689
- "Code design and performance" on page 2690

The system is getting progressively slower

Follow this guidance to resolve the problem of a slow system.

Procedure

- **Scenario:** The system is getting progressively slower.
- **Explanation:** When a process ends abnormally, an entry is made in the local error log, and a dump data file might be written to the errors or z/OS log directory. This directory is unbounded, and if you do not clear it of unwanted files, you might find that your system performance degrades due to significant space being used by old abend files.
- **Solution:** Periodically clear the errors or z/OS log directory of unwanted files.

  On Windows, use the **-w** parameter of the **mqsicreatebroker** command to create the errors directory in a hard disk drive partition that does not contain IBM App Connect Enterprise or Windows itself.

You experience configuration problems with multiple components

Follow this guidance to resolve configuration problems with multiple components.

Procedure

- **Scenario:** You are experiencing configuration problems with many components.
- **Explanation:** If you are running IBM App Connect Enterprise with many configured components, the memory footprint of the integration node processes (particularly the integration servers or DataFlowEngines) might exceed their memory limits. In particular, the user process limit might be exceeded, or the address space limit might be reached.

  You might encounter problems, such as the BIP2106E or BIP2137E error messages, when running an integration node with:

  – Many message flows
  – Multiple databases
  – Large input or output messages
- **Solution:** Use tools that are specific to your operating system to check the maximum size of the failing process, then check for any user limits (if applicable) or computer limits on process size.

  On Windows, the maximum process size is 2 GB. Increasing user limits beyond this value does not make any difference. If your integration node processes regularly reach this size, consider spreading your message flows across more integration servers to reduce the size of each one below these limits.

Message flow performance is reduced when you access message trees with many repeating records

Follow this guidance to resolve the problem of reduced performance in message trees that have repeating records.

Procedure

- **Scenario:** Message flow performance is reduced when the following conditions are true:
  - You are using ESQL processing to manipulate a large message tree.
  - The message tree consists of repeating records or many fields.
  - You have used explicit SET statements with field reference paths to access or create the fields.
  - You have observed a gradual slowing of message flow processing as the ESQL processes more fields or repetitions.
- **Explanation:** This problem occurs when you use field references, rather than reference variables, to access or create consecutive fields or records.

  Consider the following example, in which independent SET statements use field reference paths to manipulate the message tree. The SET statement takes a source and target parameter, where either or both parameters are field references:

  ```
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field = '1';
  ```

  The problem arises when the SET statement is used to create many more fields, as shown in the following example:

  ```
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field2 = '2';
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field3 = '3';
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field4 = '4';
  SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field5 = '5';
  ```

  In this example, the five fields that are created are all children of ParentA. Before the specified field can be created or modified, the integration node must navigate the named message tree to locate the point in the message tree that is to be altered. For example:

  - To access field 1, the SET statement navigates to ParentA, then to the first field, therefore involving two navigations.
  - To access field 5, the SET statement navigates to ParentA, then traverses each of the previous fields until it reaches field 5, therefore involving six navigations.

  Navigating over all the fields that precede the specified field causes the loss in performance.

  Now consider a scenario that accesses repeating fields in an input message tree; for example:

  ```
  DECLARE myChar CHAR;
  DECLARE thisRecord INT 0;
  WHILE thisRecord < 10000 DO
   SET thisRecord = thisRecord + 1;
   SET myChar = InputRoot.MRM.myParent.myRepeatingRecord[thisRecord];
  ```

```
END WHILE;
```

When index notation is used, as the count increases, the processing needs to navigate over all the preceding fields to get the one it wants; that is, it has to count over the previous records to get to the one that is represented by the current indexed reference.

- When accessing InputRoot.MRM.myParent.myRepeatingRecord[1], one navigation takes place to get to the first record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[2], two navigations take place to get to the second record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[N], *N* navigations take place to get to the N-th record.

Therefore, the total number of navigations for this WHILE loop is: 1 + 2 + 3 + ... + *N*, which is not linear.

- **Solution:** If you are accessing or creating consecutive fields or records, use reference variables. When you use reference variables, the statement navigates to the main parent, which maintains a pointer to the field in the message tree. The following example shows the ESQL that can be used to reduce the number of navigations when creating new output message tree fields:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
DECLARE outRef REFERENCE TO OutputRoot.XMLNS.TestCase.StructureA.ParentA;
SET outRef.field2 = '2';
SET outRef.field3 = '3';
SET outRef.field4 = '4';
SET outRef.field5 = '5';
```

When referencing repeating input message tree fields, you could use the following ESQL:

```
DECLARE myChar CHAR;
DECLARE inputRef REFERENCE TO InputRoot.MRM.myParent.myRepeatingRecord[1];
WHILE LASTMOVE(inputRef) DO
 SET myChar = inputRef;
 MOVE inputRef NEXTSIBLING NAME 'myRepeatingRecord';
END WHILE;
```

For further information, see “Creating dynamic field references” on page 1639.

You are experiencing poor startup times for the IBM App Connect Enterprise Toolkit

Follow this guidance to resolve the problem of poor startup times for the IBM App Connect Enterprise Toolkit.

Procedure

- **Scenario:** You are experiencing poor startup times for the IBM App Connect Enterprise Toolkit.
- **Explanation:** Poor startup times in the IBM App Connect Enterprise Toolkit can occur if you install all of the language packs.
- **Solution:** Install only those language packs that you require.

You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large projects

Follow this guidance to resolve the problem of reduced performance when you are working with large projects.

Procedure

- **Scenario:** You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large or complex projects.

- **Explanation:** Performance is reduced because of frequent project changes, such as adding and removing projects, or using **Project** > **Clean**. Complete project updates use large amounts of memory due to the size, number, and connections between files.
- **Solution:** Increase your system memory.

Performance is reduced when you run web services with small message sizes

Follow this guidance to resolve the problem of reduced performance when you run web services with small message sizes.

Procedure

- **Scenario:** You see poor response times and throughput rates when you run web services using HTTP, and send smaller messages sizes (typically less than 32 KB). Throughput rates can fluctuate with message size. IBM App Connect Enterprise running on the AIX platform might be affected.
- **Explanation:** The default configuration of HTTP enables the Nagle algorithm, which seeks to improve the efficiency of Internet Protocol networks by reducing the number of packets sent. It works by buffering small packets together, creating a smaller number of large packets. By default, the **`tcpnodelay`** setting on the sockets of the HTTPRequest is `true`. You can disable the Nagle algorithm at either the operating system level (system wide) or through IBM App Connect Enterprise (affecting only the IBM App Connect Enterprise HTTP sockets).
- **Solution:** Stop the integration node and use the following commands to disable the Nagle algorithm:

  HTTPRequest, SOAPRequest, and SCARequest nodes

  ```
  mqsichangeproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -n tcpNoDelay -v true|false -f
  mqsichangeproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -n tcpNoDelaySSL -v true|false -f
  ```

  To determine the value set, take the following steps:

  Report property values

  Start the integration node and use the following commands:

  ```
  mqsireportproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -r
  mqsireportproperties integrationNodeName -e integrationServerName
        -o HTTPConnector -r
  mqsireportproperties integrationNodeName -e integrationServerName
        -o HTTPSConnector -r
  mqsireportproperties integrationNodeName -b NodeHttpListener
        -o HTTPConnector -r
  mqsireportproperties integrationNodeName -b NodeHttpListener
        -o HTTPSConnector -r
  ```

The PutTime reported by IBM MQ on z/OS, and other times or timestamps are inconsistent

Follow this guidance to resolve the problem of inconsistently reported times and timestamps.

Procedure

- **Scenario:** The PutTime reported by IBM MQ on z/OS, and other times or timestamps are inconsistent. A difference of approximately 20 seconds is detected in:
  - Traces (including those obtained from the Trace node)
  - The MQPUTTIME timestamp in the message MQMD header
  - Timestamps obtained from ESQL (for example, in a Compute node)
- **Explanation:** IBM App Connect Enterprise reports the time using standard methods from the operating system, which do not account for leap seconds. However, on z/OS, the message putTime

that is reported by IBM MQ in the MQMD header of a message *can* account for leap seconds, using the value specified for the number of leap seconds in the CVT field.

This inconsistency can cause:

- Problems when debugging
- Problems with message flows if you use timestamps to control the flow of messages
- Misinformation

- **Solution:** Configure your system to synchronize with an NTP server, such that the CVT field is no longer relevant.

  Alternatively, add an offset to adjust a z/OS timestamp reading. For example, add 20 seconds when getting the CURRENT_TIME in ESQL.

You are experiencing reduced Java performance, or Java performance degrades after debugging a message flow

Follow this guidance to resolve the problem of reduced Java performance after debugging a message flow.

Procedure

- **Scenario:** The Java code in the JavaCompute, the Java user-defined node, or the XSLTransform node does not run as quickly as expected, or performance in these nodes degrades after debugging a message flow.

  This performance degradation might particularly affect the deployment of XML schemas. Other Java based nodes might also experience degraded performance; for example, adapter nodes (SAP, PeopleSoft, JD Edwards, Siebel), SOAP nodes, CORBA nodes, IMS nodes, CICS nodes and JMS nodes.
- **Explanation:** The integration node disables the Java JIT (just-in-time) compiler on the Java virtual machine (JVM) that is created when the integration server is started if you are debugging a message flow. A disabled Java JIT compiler provides a greater choice of debugging options, however any optimizations typically performed during JIT compilation are prevented, which might result in degraded performance.
- **Solution:** If you are not debugging a message flow:

  1. Set the JVM debug port to zero using this command:

     ```
     mqsichangeproperties integration_node_name -e integration_server_name
     -o ComIbmJVMManager -n jvmDebugPort -v 0
     ```

     For example:

     ```
     mqsichangeproperties TEST -e default
     -o ComIbmJVMManager -n jvmDebugPort -v 0
     ```

  2. Disable flow debugging.

Message flow monitoring

You can configure monitoring events on message flows, and these events can be published to IBM MQ or MQTT, to enable subscribers to receive them. You can also configure message flows to send monitoring events to a Logstash input in an Elasticsearch, Logstash, and Kibana (ELK) stack.

About this task

You can configure monitoring events on message flows, and then monitor the events as they are processed by the flows, as described in “Message flow monitoring overview” on page 2772.

You can also monitor business transactions, by creating business transaction definitions and then viewing the status of the business transactions in the web user interface. For more information, see "Monitoring business transactions" on page 2817 and "Business transaction monitoring overview" on page 2818.

Message flow monitoring overview

You can monitor message flows, learn about monitoring basics and scenarios, and decide how to configure event sources.

A message flow can be configured to publish an event when something interesting happens. The events can be published to IBM MQ or MQTT, to enable subscribers to receive them. You can also configure a message flow to send monitoring events to a Logstash input in an Elasticsearch, Logstash, and Kibana (ELK) stack.

An event is a message that contains information about the source of the event, the time of the event, and the reason for the event. The event can include the message bit stream, and can also include selected elements from the message body. These fields can be used to correlate messages that belong to the same transaction, or to convey business data to a monitoring application. You can then use these events to monitor your message flows.

You can configure events by using either of the following methods:

- Use the Flow Editor in the IBM App Connect Enterprise Toolkit to configure the message flow monitoring events.
- Use a Monitoring profile to define monitoring events that can be applied to one or more message flows that are deployed to an integration server. A Monitoring profile is a special XML file that is stored and deployed in a Policy project.

The following topics describe the concepts behind monitoring and explain how to configure monitoring for your message flows.

- "Monitoring basics" on page 2772
- "Monitoring properties and monitoring profiles" on page 2786
- "Configuring monitoring for message flows" on page 2788

For information about using business events to monitor business transactions through one or more message flows, see "Business transaction monitoring overview" on page 2818.

Monitoring basics

You can configure message flows to emit events, which can then be read and used by other applications for transaction monitoring and auditing. Monitoring events are also used by the built-in business transaction monitoring capability that is provided by IBM App Connect Enterprise, and by the record and replay capability.

The following sections introduce the concepts of monitoring:

- "Monitoring events" on page 2772
- "Event sources" on page 2773
- "Transaction events" on page 2774
- "Event output options" on page 2774
- "Publishing monitoring events" on page 2774

For more information about business transaction monitoring, see "Monitoring business transactions" on page 2817 and "Business transaction monitoring overview" on page 2818.

Monitoring events

A monitoring event is a message that is published when a message flow that is configured with a monitoring event completes the source action on which the event is configured. Monitoring events can be configured on two source types:

- Transaction, including Transaction Started, Transaction Completed.
- Terminal, when a message flow node terminal propagates.

Monitoring event messages that are published MQ or MQTT are in the form of XML or JSON documents that conform to the monitoring event schema. Monitoring event messages that are sent to a Logstash input in an Elasticsearch, Logstash, and Kibana (ELK) stack are emitted in JSON format.

Each event contains the following information:

- The source of the event.
- The name of the event.
- A sequence number and creation time.
- A correlation ID for events that are emitted by the same transaction or unit of work.

Additionally, a monitoring event can contain the following items:

- Application data that is extracted from the message.
- Part or all of the message bit stream.

You can create a message model for the monitoring event by completing the following steps in the IBM App Connect Enterprise Toolkit:

1. Click **File**.
2. Click **New**.
3. Select **Message Model**.
4. Select **IBM supplied**.
5. Click **Next**.
6. Select **Create an XML schema file by importing an IBM supplied definition**.
7. Click **Next**.
8. Select an existing Application or Library, or create a new one.
9. Select **IBM App Connect Enterprise Monitoring Event** from the list of IBM supplied messages.
10. Click **Finish**.

You can also find the monitoring event schema at: *install_location*/server/sample/Monitoring/MonitoringEventV2.xsd or *install_location*/server/sample/Monitoring/MonitoringEventV1.json.

Event sources

A message flow can emit two kinds of event:

Transaction events
Transaction events are emitted from input nodes only.

Terminal events
Terminal events are emitted from any terminal of any node, including input nodes.

An individual message flow can emit transaction events, terminal events, or both kinds of event. You can configure, enable, and disable both types of event by using the monitoring properties of the message flow, or by using a monitoring profile.

An event source address identifies an event source in a message flow.

Terminal events can be emitted from any node in a message flow. Therefore, these events can be used as an alternative to dedicated event-emitting nodes or subflows, such as the example supplied in SupportPac IA9V.

Event sources emit events only if monitoring is activated for the message flow.

Transaction events

Each input node in a message flow contains three events sources, in addition to terminal events.

Event source	Event source address	Description
Transaction start	*Nodename*.transaction.Start	The event is emitted when the message is read from the transport.
Transaction end	*Nodename*.transaction.End	The event is emitted when IBM App Connect Enterprise completes all processing of the message.
Transaction rollback	*Nodename*.transaction.Rollback	The event is emitted instead of transaction end if the message flow issues an exception that is not caught and processed in the flow.

Events are emitted subject to the evaluation of the `eventFilter` expression, as described in "Event output options" on page 2774.

If a message flow handles its own exceptions, a transaction end event is issued instead of a transaction rollback event because the flow takes control of the error and terminates normally. In this case, if you need to distinguish errors, you can configure terminal events at appropriate nodes in the flow.

Terminal events

Any terminal in a message flow can be an event source. If the event source is active, it emits an event each time a message passes through the terminal, subject to the evaluation of the `eventFilter` expression. This expression is described in "Event output options" on page 2774.

Event output options

When you configure an event source, you can define a filter to control whether the event is emitted. You can tailor event emission to your business requirements by filtering out events that do not match a set of rules. For example, you might decide to emit events only for transactions over a minimum amount:

```
$Body/StockTrade/Details/Value > 10000
```

The use of a filter can reduce the number of events that are emitted, and reduce the workload on your monitoring application.

You might filter out the transaction.Start and transaction.End events that are emitted by the MQInput node for IBM MQ Backout transaction events after a backout threshold is reached. An event monitoring application can then collect appropriate data.

```
3 >= $Root/MQMD/BackoutCount
```

Publishing monitoring events

Events are published to a topic, where they can be read by multiple subscribers. They can be published to IBM MQ or MQTT to enable subscribers to receive them. The topic name has the following structure:

- For events published on the IBM MQ pub/sub broker:
  - XML:

    ```
    $SYS/Broker/integrationNodeName/Monitoring/integrationServerName/flow_name
    ```

  - JSON:

    ```
    $SYS/Broker/integrationNodeName/MonitoringEvents/JSON/integrationServerName/flow_name
    ```

- For events published on the MQTT pub/sub broker:
  - XML:

    ```
    IBM/IntegrationBus/integrationNodeName/Monitoring/integrationServerName/flow_name
    ```

  - JSON:

    ```
    IBM/IntegrationBus/integrationNodeName/MonitoringEvents/JSON/integrationServerName/flow_name
    ```

Events can also be written to the file system in rotatable log files. The monitoring events are written to the *workDir*/Common/MonitoringEvents subdirectory.

Monitoring events can also be published to a Logstash input plug-in in an Elasticsearch, Logstash, and Kibana (ELK) stack, so that you can display the reported information in a Kibana dashboard. For more information, see "Reporting logs and monitoring events to a Logstash input in an ELK stack" on page 2869.

For more information about configuring the publication of event messages, see "Configuring the publication of event messages" on page 2807.

The hierarchical structure allows subscribers to filter the events that they receive. One subscriber can receive events from all message flows in the integration node, while another receives only the events from a single integration server. For more information about publishing and subscribing to events, see "Publish/subscribe overview" on page 1236.

You decide whether events participate in transactions when you configure a monitoring event source:

- If you want an event to be emitted only if the message flow transaction commits, configure the event source to coordinate the events with the message flow transaction.
- If you want an event to be emitted regardless of whether the message flow transaction commits or rolls back, configure the event source to emit events out of sync point. Such events are available immediately.
- A group of events can be emitted together regardless of whether the message flow transaction commits or rolls back. If you want this option, you must configure the event source to emit events in an independent unit of work.

Note:

The monitoring event is not emitted if a parser exception occurs in the following situations:

- Constructing the monitoring event from the Event Name, Event Filter, or Event Payload.
- Including the bit stream in the payload.

In these situations, the parser exception is thrown again and the message is rolled back.

Monitoring profiles

To customize events after a message flow is deployed, but without redeploying the flow, you can use a monitoring profile.

Before you can use a monitoring profile, you must deploy it to a policy project and then activate the monitoring profile on the message flow, application, or integration server. Activate the monitoring profile by using either the command or the administration REST API.

You can also configure a default monitoring profile for an integration server, by setting properties in the Defaults section of the server.conf.yaml file. This profile is then used if a message flow has no monitoring events that are configured and no specific monitoring profile applied. For example:

```
Defaults:
    monitoringProfile: 'myIS1MonitoringProfile'       # Default Monitoring profile
```

A monitoring profile is an XML document that specifies the event sources in a message flow that emits events, and the properties of those events. The monitoring profile XML must conform to XML schema file MonitoringProfile.xsd, which you can find in the samples directory of the

IBM App Connect Enterprise installation (*Install_root*/server/sample/Monitoring/MonitoringProfile.xsd). The XML file must have an extension of .monprofile.xml (for example, *monitoringProfileName*.monprofile.xml) and must be contained in a policy project.

The monitoring profile must be deployed before you start the message flows that use it. If a message flow is configured to use a monitoring profile that is not deployed, a warning is issued. You must manually stop and start the message flow after the monitoring profile is deployed in order for it to be used by the message flow. If you redeploy a monitoring profile, all message flows that use that monitoring profile are stopped and restarted to use the new values.

You can attach, change, or detach monitoring profiles from a message flow by using the **mqsichangeflowmonitoring** command. For more information, see "Applying and activating a monitoring profile" on page 2798.

The following example of a monitoring profile XML document contains a single event source to illustrate the structure.

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
    <p:eventSource p:enabled="true" p:eventSourceAddress="SOAPInput.transaction.Start">
        <p:eventPointDataQuery>
            <p:eventIdentity>
                <p:eventName p:literal="" p:queryText=""/>
            </p:eventIdentity>
            <p:eventCorrelation>
                <p:localTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:parentTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:globalTransactionId p:queryText="" p:sourceOfId="automatic"/>
            </p:eventCorrelation>
           <p:eventFilter p:queryText="true()"/>
           <p:eventUOW p:unitOfWork="messageFlow" />
        </p:eventPointDataQuery>
        <p:applicationDataQuery>
            <p:simpleContent p:dataType="boolean" p:name="" p:targetNamespace="">
                <p:valueQuery p:queryText=""/>
            </p:simpleContent>
            <p:complexContent p:name="">
                <p:payloadQuery p:queryText=""/>
            </p:complexContent>
        </p:applicationDataQuery>
        <p:bitstreamDataQuery p:bitstreamContent="all" p:encoding="base64Binary"/>
    </p:eventSource>
</p:monitoringProfile>
```

The root element is p:monitoringProfile. It contains one or more p:eventSource elements, each of which specifies an event source and defines its properties. Each p:eventSource element contains:

- A p:eventPointDataQuery element that provides key information about the event.
- Optional: A p:applicationDataQuery element if the event payload includes data fields that are extracted from a message.
- Optional: A p:bitstreamDataQuery element if the event payload includes bitstream data from a message.

Creating a monitoring profile

If you have a deployed message flow with monitoring properties that were configured by using the Message Flow editor in the IBM App Connect Enterprise Toolkit, you can use either the command or the administration REST API to extract those monitoring properties from the deployed flow and create the equivalent monitoring profile (.monprofile.xml file) for the message flow. You can then use this profile as a starting point for creating other monitoring profiles.

Alternatively, you can create the .monprofile.xml file manually, by following the steps described in "Creating a monitoring profile" on page 2793.

XPath queries and XML namespaces

If an XPath query contains a component that has an XML namespace, the XPath contains a namespace prefix for the namespace. The namespace prefix in all `prefixMapping` elements in a monitoring profile must be unique. For example, the following XPath refers to components in two different namespaces:

```
<p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID" />
```

For the integration node to resolve the namespace prefix, the namespace URL must also be provided. Supply a `prefixMapping` element for each namespace:

```
      <p:localTransactionId p:sourceOfId="query" p:queryText="$Body/soapenv:Header/wsa:messageID">
        <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope" />
        <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing" />
      </p:localTransactionId>
```

Monitoring profile examples

The following XML documents conform to the monitoring profile schema.

Monitoring profile 1: Two event sources, each supplying an event name

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP start event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.End">
  <p:eventPointDataQuery>
    <p:eventIdentity>
      <p:eventName p:literal="SOAP end event"/>
    </p:eventIdentity>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 2: Supply an alternative local correlator

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
 <p:eventSource p:eventSourceAddress="SOAPInput.transaction.Start">
  <p:eventPointDataQuery>
    <p:eventCorrelation>
      <p:localTransactionId p:queryText="$Body/soapenv:Header/wsa:messageID" p:sourceOfId="query">
        <p:prefixMapping p:prefix="soapenv" p:URI="http://www.w3.org/2003/05/soap-envelope"/>
        <p:prefixMapping p:prefix="wsa" p:URI="http://www.w3.org/2005/08/addressing"/>
      </p:localTransactionId>
    </p:eventCorrelation>
  </p:eventPointDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 3: Include two simple fields from the message

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
  <p:applicationDataQuery>
    <p:simpleContent p:dataType="integer" p:name="InvoiceNumber">
      <p:valueQuery p:queryText="$Body/invoice/invoiceNo"/>
    </p:simpleContent>
    <p:simpleContent p:dataType="string" p:name="BatchID">
      <p:valueQuery p:queryText="$Body/batch/batchNo"/>
    </p:simpleContent>
```

```
  </p:applicationDataQuery>
 </p:eventSource>
</p:monitoringProfile>
```

Monitoring profile 4: Include the bitstream, encoded as CDATA

By default, bitstreams are encoded in base64Binary format. The following monitoring profile changes the encoding to CDATA.

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
 <p:eventSource p:eventSourceAddress="MQInput.terminal.out">
    <p:bitstreamDataQuery p:bitstreamContent="body" p:encoding="CDATA"/>
 </p:eventSource>
</p:monitoringProfile>
```

CDATA encoding is not suitable for all types of data. Use CDATA only when `@p:bitstreamContent="body"`. Do not use CDATA if your message bitstreams might contain characters that are not allowed in XML (see http://www.w3.org/TR/2006/REC-xml-20060816/#charsets).

Monitoring profile 5: One *eventSource* that applies to all transaction events for a message flow and includes the available bitstream as *base64binary*.

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/prod/websphere/messagebroker/10.0.0.0/monitoring/profile"
 p:version="2.0">
    <p:eventSource p:enabled="true" p:eventSourceAddress="transaction.all">
        <p:eventPointDataQuery>
            <p:eventCorrelation>
                <p:localTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:parentTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:globalTransactionId p:queryText="" p:sourceOfId="automatic"/>
            </p:eventCorrelation>
        </p:eventPointDataQuery>
        <p:bitstreamDataQuery p:bitstreamContent="all" p:encoding="base64Binary"/>
    </p:eventSource>
</p:monitoringProfile>
```

The monitoring event

You can configure IBM App Connect Enterprise to emit a monitoring event, in XML or JSON format, when something significant happens. Events are typically emitted to support transaction monitoring and auditing, and business transaction monitoring.

You can configure the source of the event and the content of the event by using either the monitoring properties of a message flow or a monitoring profile.

Each event contains the following information:

- Source of the event.
- Name of the event.
- Creation time and sequence number.
- Correlation ID for events that are emitted by the same transaction or unit of work.
- Details of the message flow.

A terminal emits an event only if the message passes through that terminal. In particular, the output terminal of an output node emits events only if it is connected.

A monitoring event can also contain the following items:

- Application data extracted from the message.
- Part or all of the message bit stream. All nodes can produce bit streams, which can be included in monitoring events.

You can configure monitoring events to be formatted in XML or JSON format. By default, event XML in IBM App Connect Enterprise conforms to the monitoring event schema `MonitoringEventV2.xsd`. Alternatively, you can configure monitoring events to be formatted in JSON format, as defined by the

MonitoringEventV1.json schema. For compatibility with IBM Integration Bus Version 10.0, you can configure monitoring events to use the WMBEvent.xsd schema.

For information about the monitoring event formats, see the following sections:

- “XML MonitoringEventV2 format” on page 2779
- “JSON MonitoringEventV1” on page 2779
- “XML WMB format” on page 2780

For information about the sequencing of events and the default values that are assigned to monitoring events, see “Event sequencing” on page 2781 and “Default content of monitoring events” on page 2781.

For information about configuring monitoring events that contain complex content, see “Including complex content in the event payload of a monitoring event ” on page 2802.

For more information about how to configure monitoring events to be emitted and formatted according to these schema definitions, see “Activating monitoring” on page 2804.

XML MonitoringEventV2 format

The XML schema definition file is MonitoringEventV2.xsd, which is provided in the server/sample/Monitoring subdirectory of your IBM App Connect Enterprise installation directory.

To access the schema from the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Message Model**. Select **IBM supplied** . Click **Next**, then select **Create an XML file by importing an IBM supplied definition** . Click **Next**, and then select **IBM App Connect Enterprise Monitoring Event**.

The following example shows a monitoring event in XML MonitoringEventV2 format:

```
<mon:event xmlns:mon="http://www.ibm.com/xmlns/monitoring/event/v2">
 <mon:eventPointData>
  <mon:eventData mon:productVersion="110010"
   mon:profileVersion="11" mon:eventSourceAddress="HTTP Input.transaction.Start">
   <mon:eventIdentity mon:eventName="HTTP Input.TransactionStart" />
   <mon:eventSequence mon:creationTime="2020-07-20T16:20:21.776758Z"
    mon:counter="1" />
   <mon:eventCorrelation mon:localTransactionId="40ddc412-f102-4840-a212-8b793d12b6bf-1"
    mon:parentTransactionId="" mon:globalTransactionId="" />
  </mon:eventData>
  <mon:messageFlowData>
   <mon:integrationServer mon:name="110010"
    mon:hostName="DESKTOP-7Q9JIP6" />
   <mon:application mon:name="Transformation_Map" />
   <mon:messageFlow mon:uniqueFlowName="110010.Transformation_Map.Transformation_Map"
    mon:name="Transformation_Map" mon:threadId="12244" />
   <mon:node mon:nodeLabel="HTTP Input" mon:nodeType="ComIbmWSInputNode" />
  </mon:messageFlowData>
 </mon:eventPointData>
 <mon:bitstreamData>
  <mon:bitstream mon:serializationEncoding="base64Binary"
   mon:dataCCSID="1208"
 mon:dataEncoding="546">UE9TVCBodHRwOi8vbG9jYWxob3N0Ojc4MDAvVHJhbnNmb3JtYXRpb25fTWFwIEhUVFAvMS4xDQpDb250ZW50LVR5cGU6
Pg0KPFNhbGVFbnZlbG9wZT4NCgk8SGVhZGVyPg0KCQk8U2FsZUxpc3RDb3VudD4xPC9TYWxlTGlzdENvdW50Pg0KCQk8VHJhbnNmb3JtYXRpb25UeXBl
+DQoJCQk8SW5pdGlhbD5UPC9Jbml0aWFsPg0KCQkJPEluaXRpYWw
+SjwvSW5pdGlhbD4NCgkJCTxTdXJuYW1lPkR1bm53aW48L1N1cm5hbWU
+DQoJCQk8SXRlbT4NCgkJCQk8Q29kZT4wNDwvQ29kZT4NCgkJCQk8Q29kZT4wNTwvQ29kZT4NCgkJCQk8Q29kZT4wMTwvQ29kZT4NCgkJCQk8RGVzY3J
+Qm9va3MgYW5kIE1lZGlhPC9DYXRlZ29yeT4NCgkJCQk8UHJpY2U
+MjIuMzQ8L1ByaWNlPg0KCQkJCTxRdWFudGl0eT4wMjwvUXVhbnRpdHk
+DQoJCQk8L0l0ZW0+DQoJCQk8QmFsYW5jZT44MS44NDwvQmFsYW5jZT4NCgkJCTxDdXJyZW5jeT5FdXJvczwvQ3VycmVuY3k
+DQoJCTwvSW52b2ljZT4NCgk8L1NhbGVMaXN0Pg0KCTxUcmFpbGVyPg0KCQk8Q29tcGxldGlvblRpbWU
+MTIuMDAuMDA8L0NvbXBsZXRpb25UaW1lPg0KCTwvVHJhaWxlcj4NCjwvU2FsZUVudmVsb3BlPg==</mon:bitstream>
 </mon:bitstreamData>
</mon:event>
```

JSON MonitoringEventV1

The JSON schema definition file is MonitoringEventV1.json, which is provided in the server/sample/Monitoring subdirectory of your IBM App Connect Enterprise installation directory.

The following example shows a monitoring event in JSON MonitoringEventV1 format:

```
{
 "type": "ace_monitoring_event",
 "ibm_product": "IBM App Connect Enterprise",
 "ibm_recordtype": "monitor",
 "host": "DESKTOP-7Q9JIP6",
 "module": "integration_server.test",
 "ibm_serverName": "test",
 "productVersion": "12020",
 "profileVersion": "11",
 "eventSourceAddress": "Compute.terminal.out",
 "eventName": "MainFlow.Compute.OutTerminal",
 "ibm_datetime": "2021-09-08T15:56:29.046935Z",
 "localTransactionId": "e9e832b3-0a92-464c-b7f1-364547160f06-4",
 "parentTransactionId": "",
 "globalTransactionId": "",
 "applicationName": "MonitorEventsApp",
 "messageFlowUniqueName": "integration_server.test.MonitorEventsApp.MonEvents",
 "messageFlowName": "MonEvents",
 "messageFlowNodeName": "Compute",
 "messageFlowNodeType": "ComIbmComputeNode",
 "messageFlowNodeTerminalName": "out",
 "applicationData": {
  "complexContent": [{
   "elementName": "Msg",
   "elementValue": {
    "bitstream": {
     "Msg": {
      "mainflow": "MonEvents"
     }
    }
   }
  }]
 },
 "bitstreamData": {
  "bitstream":
 "504f535420687474703a2f2f6c6f63616c686f73743a373830302f4d6f6e4576656e747320485454502f312e310d0a486f73743a206c6f6361
  "serializationEncoding": "hexBinary",
  "dataCCSID": "1208",
  "dataEncoding": "546"
 },
 "ibm_sequence": "1"
}
```


XML WMB format

The XML schema definition file is `WMBEvent.xsd`, which is provided in the `server/sample/Monitoring` subdirectory of your IBM App Connect Enterprise installation directory.

To access the schema from the IBM App Connect Enterprise Toolkit, click **File** > **New** > **Message Model**. Select **IBM supplied**. Click **Next**, then select **Create an XML file by importing an IBM supplied definition** . Click **Next**, and then select **IBM Integration Bus Monitoring Event**.

The following example shows a monitoring event in XML WMB format:


```
<wmb:event
 xmlns:wmb="http://www.ibm.com/xmlns/prod/websphere/messagebroker/6.1.0/monitoring/event">
 <wmb:eventPointData>
  <wmb:eventData wmb:productVersion="110010"
   wmb:eventSchemaVersion="11" wmb:eventSourceAddress="HTTP Input.transaction.Start">
   <wmb:eventIdentity wmb:eventName="HTTP Input.TransactionStart" />
   <wmb:eventSequence wmb:creationTime="2020-07-20T16:15:24.728446Z"
    wmb:counter="1" />
   <wmb:eventCorrelation wmb:localTransactionId="c4b687e6-4f51-4f61-ae3b-92c3115e9e42-1"
    wmb:parentTransactionId="" wmb:globalTransactionId="" />
  </wmb:eventData>
  <wmb:messageFlowData>
   <wmb:broker wmb:name="integration_server" />
   <wmb:executionGroup wmb:name="110010" />
   <wmb:messageFlow
    wmb:uniqueFlowName="integration_server.110010.Transformation_Map.Transformation_Map"
    wmb:name="Transformation_Map" wmb:threadId="17376" />
   <wmb:node wmb:nodeLabel="HTTP Input" wmb:nodeType="ComIbmWSInputNode" />
  </wmb:messageFlowData>
 </wmb:eventPointData>
 <wmb:bitstreamData>
```

```
  <wmb:bitstream
 wmb:encoding="base64Binary">UE9TVCBodHRwOi8vbG9jYWxob3N0Ojc4MDAvVHJhbnNmb3JtYXRpb25fTWFwIEhUVFAvMS4xDQpDb250ZW50LVR
Pg0KPFNhbGVFbnZlbG9wZT4NCgk8SGVhZGVyPg0KCQk8U2FsZUxpc3RDb3VudD4xPC9TYWxlTGlzdENvdW50Pg0KCQk8VHJhbnNmb3JtYXRpb25UeXBl
+DQoJCQk8SW5pdGlhbD5UPC9Jbml0aWFsPg0KCQkJPElluaXRpYWw
+SjwvSW5pdGlhbD4NCgkJCTxTdXJuYW1lPkR1bm53aW48L1N1cm5hbWU
+DQoJCQk8SXRlbT4NCgkJCQk8Q29kZT4wNDwvQ29kZT4NCgkJCQk8Q29kZT4wNTwvQ29kZT4NCgkJCQk8Q29kZT4wMTwvQ29kZT4NCgkJCQk8RGVzY3Jp
+Qm9va3MgYW5kIE1lZGlhPC9DYXRlZ29yeT4NCgkJCQ0UHJpY2U
+MjIuMzQ8L1ByaWNlPg0KCQkJCTxRdWFudGl0eT4wMjwvUXVhbnRpdHk
+DQoJCQk8L0l0ZW0+DQoJCQk8QmFsYW5jZT44MS44NDwvQmFsYW5jZT4NCgkJCTxDdXJyZW5jeT5FdXJvczwvQ3VycmVuY3k
+DQoJCTwvSW52b2ljZT4NCgk8L1NhbGVMaXN0Pg0KCTxUcmFpbGVyPg0KCTxUcmFpbGVyU2FsZUVudmlsb3BlPg==</wmb:bitstream>
 </wmb:bitstreamData>
</wmb:event>
```

Event sequencing

To enable event-processing software to sequence the events correctly, both an ISO 8601 timestamp and a counter attribute are produced. The counter attribute is in the EventSequence element of the monitoring event. This counter starts at 1 for the first event that is produced by the processing of a message (typically the transaction.Start event) and is incremented for each subsequent event produced. It is reset to 1 at the start of the next message. The creation time and counter are always produced on all monitoring events. The software that processes the events can choose which field, or a combination of the two, to use to sequence the events.

If a message is processed successfully, the monitoring events that are generated have a contiguous, incrementing set of counter values. The value starts at 1 and is assigned at the time when the event is created. If a message fails and is rolled back, gaps can occur in the counter sequence. The missing values are the values that were used for events that were produced as part of the message flow unit of work that was rolled back.

Monitoring events can be included in the main unit of work of a message flow, an independent unit of work, or outside of a unit of work. Therefore, if a message flow fails, the sequence numbers are not necessarily contiguous. For example, consider the following scenario:

- Sequence 1. The transaction start event.
- Sequence 2, 3, 4. Events in the message flow unit of work.
- Sequence 5. The independent unit of work.
- Sequence 6. An event outside of a unit of work.
- Sequence 7. An event in the message flow unit of work.
- Sequence 8. An event outside of a unit of work.
- The flow then fails and is rolled back.
- Sequence 9. The transaction rollback event in the independent unit of work.

Only sequence numbers 1, 5, 6, 8 and 9 are sent to the monitoring application.

Default content of monitoring events

When an event is emitted, the fields for the event are created by using the information that is provided by the monitoring properties of the message flow or the monitoring profile. Any event field that is not explicitly specified is given a default value, as shown in the following table:

Field in event	Default value MonitoringEventV2	Default Value WMB
bitstreamData.bitstream@wmb:codedCharSetId	The coded character set identifier of the message data provided in the bitstream	Not applicable
bitstreamData.bitstream@wmb:encoding	The data encoding of the message data provided in the bitstream	Not applicable
eventData/@wmb:eventSourceAddress	No default; you must provide this information.	No default; you must provide this information.

Field in event	Default value MonitoringEventV2	Default Value WMB
eventData/@wmb: eventSchemaVersion	Not applicable	`6.1.0.3`
eventData/ @wmb:productVersion		`7000`
eventData/ eventIdentity/ @wmb:eventName	The default is derived from @eventSourceAddress.	The default is derived from @eventSourceAddress.
eventData/ eventSequence/ @counter	The counter is set to 1 for first event that is emitted and increased by 1 for each subsequent event.	The counter is set to 1 for first event that is emitted and increased by 1 for each subsequent event.
eventData/ eventSequence/ @creationTime	The date and time when the event was created.	The date and time when the event was created.
eventData/ eventCorrelation/ @localTransactionId	A generated unique identifier. This identifier is not generated if only transaction rollback or end events, or both, are configured, unless a transaction start event is configured as well.	A generated unique identifier. This identifier is not generated if only transaction rollback or end events, or both, are configured, unless a transaction start event is configured as well.
eventData/ eventCorrelation/ @parentTransactionId	No default. Unless you set this value, an empty string is used.	No default. Unless you set this value, an empty string is used.
eventData/ eventCorrelation/ @globalTransactionId	No default. Unless you set this value, an empty string is used.	No default. Unless you set this value, an empty string is used.
messageFlowData/ @application	Application name	Not applicable
messageFlowData/ @library	Library name	Not applicable
messageFlowData/ broker/@name	Not applicable.	The name of the integration node.
messageFlowData/ executionGroup/ @name	Not applicable.	The name of the integration server.
messageFlowData/ integrationNode/ @name	The name of the integration node.	Not applicable.
messageFlowData/ integrationServer	The name of the integration server.	Not applicable.
messageFlowData/ messageFlow/@name	The name of the message flow.	The name of the message flow.
messageFlowData/ messageFlow/ @uniqueFlowName	A string composed of the names of the integration node, integration server, and flow in the form: *integrationNodeName*`.`*integrationServerName*`.`*messageFlowName*	A string composed of the names of the integration node, integration server, and flow in the form: *integrationNodeName*`.`*integrationServerName*`.`*messageFlowName*

Field in event	Default value MonitoringEventV2	Default Value WMB
messageFlowData/ messageFlow/ @threadId	The thread ID of the message flow. The format depends on the operating system.	The thread ID of the message flow. The format depends on the operating system.
messageFlowData/ node/@nodeLabel	The label of the node that emitted the event.	The label of the node that emitted the event.
messageFlowData/ node/@nodeType	The type of the node that emitted the event.	The type of the node that emitted the event.
messageFlowData/ node/@nodeDetail	Optional information about the node. **MQInput** The name of the queue. **Other nodes** Omitted.	Optional information about the node. **MQInput** The name of the queue. **Other nodes** Omitted.
applicationData	No default; omitted if not provided.	No default; omitted if not provided.
bitstreamData	No default; omitted if not provided.	No default; omitted if not provided.

Correlation and monitoring events

A monitoring application uses correlation attributes to identify events that belong to the same business transaction.

A business transaction can be any of the following scenarios:

- A single invocation of a message flow.
- Multiple invocations of the same message flow from a parent application. The parent application might be another message flow.
- Multiple invocations of various message flows from a parent application. The parent application might be another message flow.

Three correlation attributes are available for you to use in your events: *local correlator*, *parent correlator* and *global correlator*. The exact usage of the correlation attributes varies depending on the requirements. For example, a parent application can pass its transaction identifier to the child message flow (perhaps in a header) so that the child message flow can report it in the event as a parent correlator.

Every emitted monitoring event can contain a local correlator, a parent correlator, and a global correlator. Correlation information is placed in the following attributes of the event:

```
wmb:eventPointData/wmb:eventCorrelation/@wmb:localTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:parentTransactionId
wmb:eventPointData/wmb:eventCorrelation/@wmb:globalTransactionId
```

You can specify correlation information when you configure the event.

If you do not specify any correlation information when you configure your events, no correlation attributes will be used.

If you do specify correlation information, you must configure the correlation attributes to be used, and where they will read their value from. Typically you need to specify correlation information only for the first event source in the message flow; by default all later event sources retrieve the same value from the Environment tree.

The exact steps for specifying correlation information depend on whether you are using monitoring properties or a monitoring profile to configure your events, but the principle is the same for both techniques:

Local correlator

If you want to reuse the local correlator from the Environment tree, specify `Automatic`. If no local correlator exists yet, a new unique value will be generated and saved in the Environment tree.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a correlator value unique to this invocation of the message flow. The extracted value is saved in the Environment tree as the local correlator.

Parent correlator

If you want to reuse the parent correlator from the Environment tree, specify `Automatic`. If no parent correlator exists yet, no parent correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the parent correlator. The extracted value is saved in the Environment tree as the parent correlator.

Global correlator

If you want to reuse the global correlator from the Environment tree, specify `Automatic`. If no global correlator exists yet, no global correlator will be used.

If you want to use a value contained in a location in the message, specify the location of the correlator by supplying an XPath into the message tree. Ensure that the specified location contains a suitable value for the global correlator. The extracted value is saved in the Environment tree as the global correlator.

A global correlator must be present if the event is to be used by the built-in business transaction monitoring capability of App Connect Enterprise.

When a correlator value has been set, it is saved in the Environment tree. Later event sources can reuse the saved value by specifying `Automatic`. There is no need to use the same XPath in all the event sources in your message flow, and doing so might adversely affect performance.

The locations in the Environment tree used to save correlator values for use by later events are:

```
Environment.Monitoring.EventCorrelation.localTransactionId
Environment.Monitoring.EventCorrelation.parentTransactionId
Environment.Monitoring.EventCorrelation.globalTransactionId
```

The following message tree locations often contain a value that can be used as a correlator:

```
$Root/MQMD/MsgId
$Root/MQMD/CorrelId
$Root/JMSTransport/Transport_Folders/Header_Values/JMSMessageID
$Root/JMSTransport/Transport_Folders/Header_Values/JMSCorrelationID
$LocalEnvironment/Destination/HTTP/RequestIdentifier
$LocalEnvironment/Wildcard/WildcardMatch
```

Tip: If the three available correlators are not sufficient, you can configure the event to extract other correlation fields from the message and place them in the `wmb:applicationData/wmb:simpleContent` section of the event.

Tip: The Collector node and the AggregateControl node do not preserve the Environment tree for later nodes in the message flow. If you want to use the same correlator value later in the flow, ensure that the correlator value is available in the message tree, and that the first event source after the Collector or AggregateControl node specifies the location of the correlator by supplying an XPath into the message tree.

Scenarios

Scenario 1: In this scenario, all three correlators are used to monitor data that starts in an external process. Several message flows then transform the data.

- The `globalTransactionID` field contains an identifier from the message header or payload. This identifier correlates events from the external process and IBM App Connect Enterprise.
- The `parentTransactionID` correlates events in IBM App Connect Enterprise from different message flows.
- The `localTransactionID` correlates events from the same message flow.

Scenario 2: In this scenario, the `parentTransactionID` field is used to correlate request and reply messages between two message flows:

- The Request flow sends a **`purchaseOrder`** request to an external application for processing.
- The Reply flow receives a confirmation reply from the external application when the **`purchaseOrder`** has been processed.

You need to correlate the request and replies belonging to the same purchase order. You can do this by setting the `parentTransactionID` to a field in the **`purchaseOrder`**, such as a **`purchaseOrderID`**, which is available in both the request and reply.

Example XPath expressions for event filtering

Use numeric, string, or Boolean expressions when configuring an event source, to determine whether the event is emitted.

When you configure an event source, using either monitoring properties or a monitoring profile, you use an XPath expression to determine whether the event is emitted.

- If the event is always required, use `true()`.
- If the event is required only in certain circumstances, use an expression of the form

```
xpath-query relational-operator value
```

Comparing numeric values

To emit an event only when the value is greater than 10 000, for example, enter an expression such as this:

```
$Body/StockTrade[1]/Details[1]/Value[1] > 10000
```

The [1] suffixes in the query specify that the first occurrence of the element within its parent is required. If these suffixes are not specified, the XPath engine searches the message for other occurrences of each element. This search might adversely affect performance.

Comparing string values

To emit an event only when the company is "Stock Co" for example:

```
$Body/StockTrade[1]/Details[1]/Company[1] = 'Stock Co'
```

Comparing Boolean values

Consider the example of a shares transfer approval. The approval flag in the message tree is a Boolean value. You cannot simply specify the element name, because this always returns true if the element exists. Instead, you query the value of the element, and compare the value to the string 'true' to yield the actual true or false result. The XPath query is:

```
$Body/StockTrade[1]/Shares[1]/Transfer[1]/Approved[1] = 'true'
```

XPath queries that return a nodeset, such as `$Body/StockTrade[1]/Details[1]`, are always evaluated as false, because they cannot be converted to a Boolean value.

Monitoring properties and monitoring profiles

Customize the events that are produced by a message flow by using monitoring properties or a monitoring profile. To customize events when you are designing the message flow, or to apply generic monitoring events to a flow without having to configure at design time, use monitoring properties. To customize events after a message flow is deployed, but without redeploying, use a monitoring profile.

If you have a deployed message flow with monitoring properties that were configured by using the Message Flow editor in the IBM App Connect Enterprise Toolkit, you can use the command or the administration REST API to extract those properties. You can then use the extracted properties to create the equivalent monitoring profile (`.monprofile.xml`) for the message flow, and use this profile as a starting point for creating other monitoring profiles.

Monitoring properties

By using the IBM App Connect Enterprise Toolkit Message Flow editor, you can configure event sources on most nodes in a message flow. A message flow node that supports the configuration of event sources has a Monitoring tab on its Properties view. Use this tab to add events and to set their properties. When you deploy the message flow in a BAR file, the monitoring properties are included as part of the message flow.

Key facts about monitoring properties:

- Use monitoring properties when you want to configure events at message flow design time.
- Monitoring properties apply only to the message flow in question. You can share monitoring properties between message flows by creating reusable subflows.
- Monitoring properties are deployed in a BAR file as part of the message flow.
- Use the **mqsichangeflowmonitoring** command to activate a message flow, or all message flows in an application or integration server to emit monitoring events.
- To change any other properties of a monitoring event, alter the monitoring properties in the message flow editor and then save and redeploy the flow in a BAR file.
- Use the **mqsireportflowmonitoring** command to report a list of the names of the event sources for a message flow.

Monitoring profiles

When you create a monitoring profile, you deploy it to a policy project and then activate it on a specific message flow, application, or integration server. You can activate the profile by using either the command or the administration REST API. Alternatively, you can specify the monitoring profile as a BAR file override, as described in Configurable properties in a file.

Key facts about monitoring profiles:

- You can configure a default monitoring profile for an integration server, by setting properties in the `Defaults` section of the `server.conf.yaml` file. This profile is then used when a message flow has no monitoring events that are configured and no specific monitoring profile applied. For example:

```
Defaults:
    monitoringProfile: 'myIS1MonitoringProfile'       # Default Monitoring profile
```

- You can override a deployed monitoring profile by putting an updated `.monprofile.xml` file in the integration server's overrides directory, as described in “Order of precedence for overrides” on page 326.
- You can use a monitoring profile when you want to configure events for a message flow that is deployed and for which no events are configured.

- You can use a monitoring profile to override the monitoring properties of a message flow that is deployed, as an alternative to redeploying the BAR file. The monitoring profile replaces all monitoring properties.
- A single monitoring profile can be applied to many message flows. Each flow that it is applied to issues events only if the event sources that are defined in the profile match the message flow. Transaction sources always apply, but terminal sources apply only if the node names are matched.
- A monitoring profile must be deployed in a policy project. The XML file must have an extension of `.monprofile.xml`

  For more information, see “Creating a monitoring profile” on page 2793, and “Deploying and redeploying a monitoring profile” on page 2797.
- Use the command to associate the monitoring profile with a message flow, or to activate the message flow to emit monitoring events.
- Use the REST API to associate the monitoring profile with a message flow. Use a POST to make a change that applies until the message flow is restarted or redeployed. For example, for a message flow deployed in a managed integration server:

  ```
  curl -v -X POST "http://<host>:<port>/apiv2/servers/<server name>/applications/<app
   name>/messageflows/<flow name>/attach-monitoring-profile?project=<policy project
   name>&profile=<profile-name>
  ```

  Use a PATCH to have the change persisted though restart or redeploy. For example:

  ```
  curl -v -X PATCH -H "Content-Type: application/json" -d Monitoring\monProf.json "http://
  <host>:<port>/apiv2/servers/<server name>/applications/<app name>/messageflows/<flow name>
  ```

- Use the command in the following situations:
  - To report a list of event sources for a message flow.
  - To report the monitoring profile for a message flow. You can edit the profile, by adding or changing event sources, then update it by using the **mqsichangeproperties** command.
  - To extract and create the equivalent monitoring profile `monprofile.xml` file for a message flow. If the monitoring properties for the message flow were configured in the IBM App Connect Enterprise Toolkit flow editor, you can place the monitoring profile `monprofile.xml` file into a Policy Project and edit the monitoring profile with an XML editor. Then, you can add it to a BAR file and deploy it. You can then associate the new monitoring profile with the original message flow, or other suitable message flows by using the **mqsichangeflowmonitoring** command. You can use this technique to override the monitoring properties for a message flow, without having to edit the message flow in the IBM App Connect Enterprise Toolkit toolkit and redeploy the BAR file.
- Use the REST API to extract a monitoring profile. For example, for a flow in an independent integration server:

  ```
  curl -v -X POST -H "Accept: application/xml" http://<host>:<port>/apiv2/applications/<app
   name>/messageflows/<flow name>/extract-monitoring-profile
  ```

  Use a PATCH to have the change persisted though a restart or deploy. For example:

  ```
  curl -v -X PATCH -H "Content-Type: application/json" -d Monitoring\monProf.json "http://
  <host>:<port>/apiv2/servers/<server name>/applications/<app name>/messageflows/<flow name>
  ```

where the content of the `Monitoring\monProf.json` file is: `{"properties": {"monitoringProfile":"{<policy project name>}:<monitoring profile name>"}}`

Configuring monitoring for message flows

You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring.

Before you begin

- To learn about the basic concepts of monitoring, see “Monitoring basics” on page 2772.
- To learn about monitoring IBM App Connect Enterprise event messages, see “Configuring and subscribing to performance and monitoring events” on page 2807.
- To decide whether to use monitoring properties or a monitoring profile, see “Monitoring properties and monitoring profiles” on page 2786.
- If you are using the default configuration with the built-in MQTT broker, the `BusinessEvents` group, which includes monitoring events, is disabled for MQTT. Check your pub/sub broker configuration. For more information, see “Configuring the built-in MQTT pub/sub broker” on page 2811.

About this task

Note: Monitoring messages are published to IBM MQ pub/sub brokers by using the persistent as topic option. Publications resolve to be nonpersistent by default, but you can change a publication to be persistent by configuring named topics in IBM MQ. For more information, see the Subscriptions and message persistence topic in the IBM MQ product documentation online.

Monitoring messages are published to MQTT brokers as nonpersistent by default. However, subscribers to MQTT brokers specify the maximum quality of service (QoS) that they require, which determines whether the message is delivered only once or more than once, and whether a confirmation of receipt is required. For more information, see http://mosquitto.org/man/mqtt-7.html.

To receive monitoring events, complete the following tasks.

Procedure

1. Configure event sources on the flow by using either monitoring properties or a monitoring profile.
   - “Configuring monitoring event sources by using monitoring properties” on page 2788
   - “Configuring monitoring event sources by using a monitoring profile” on page 236
2. Enable event sources by using the monitoring properties on the node.
3. Ensure that event publication for monitoring messages is enabled and is correctly configured.
   For more information, see “Configuring the publication of event messages” on page 2807.
4. Subscribe to the topic for the flow; see “Subscribing to event message topics” on page 2813.
5. Optional: Optionally, you can collect monitoring messages for all message flows in a server as described in “Collecting monitoring messages for all message flows in a server” on page 2806.

Configuring monitoring event sources by using monitoring properties

In the Message Flow editor, use the **Monitoring** tab on the properties of a message flow node to add one or more monitoring events.

Before you begin

Read the following topics:

- “Monitoring basics” on page 2772
- “Configuring monitoring for message flows” on page 2788

Ensure that you have a message flow that contains a node to which you want to add a monitoring event.

You cannot use monitoring properties to configure transaction events on the following nodes:

node
node

For these nodes, use a monitoring profile instead, as described in "Configuring monitoring event sources by using a monitoring profile" on page 236.

About this task

An event source is a point in a message flow from which a monitoring event can be emitted. Each event source has a set of properties that control the contents of the monitoring events that it emits. To create an event, complete the following steps.

Procedure

1. In the Message Flow editor, select the relevant node to display the properties for that node.
2. Select the **Monitoring** tab.
3. Click **Add**.

   The **Add event** window is displayed.
4. In the **Event Source** field, select the source of the event.

   When you select the event source, the corresponding value for `Event Source Address` is displayed as a read-only property. The event source information is used to populate the attributes of the `wmb:eventPointData/wmb:messageFlowData/wmb:node` element of the event.

   Tip: If you decide to enable or disable events by using the **mqsichangeflowmonitoring** command, you must specify a value for `Event Source Address`, not `Event Name`.
5. In the **Event Name** field, select either **Literal** or **Data location**.

   Every monitoring event has a name that is placed in the `wmb:eventPointData/wmb:eventIdentity/@wmb:eventName` attribute of the event. The default names are shown in the following table:

Event source	Default event name	Example
Transaction start	*nodeLabel*.TransactionStart	MQInput.TransactionStart
Transaction end	*nodeLabel*.TransactionEnd	MQInput.TransactionEnd
Transaction rollback	*nodeLabel*.TransactionRollback	MQInput.TransactionRollback
Terminal	*nodeLabel.terminal_label*.Terminal	MQInput.OutTerminal

   You can override the default value in the following ways:

   - By specifying an alternative literal string.
   - By specifying an XPath query. The query extracts the event name from a field in the input message. Click **Edit** to use the XPath Expression Builder.
6. Optional: In the **Event Filter** section, provide an XPath expression to control whether the event is emitted.

   Type in the expression (for example, `$Body/StockTrade/Details/Value > 10000`) or click **Edit** to launch the XPath Expression Builder.

   The expression must evaluate to true or false, and can reference fields in the message tree, or elsewhere in the message assembly. The default value is `true()`, which means that the event is always produced.

7. Optional: Complete the **Event Payload** field if the event must contain selected data fields that are extracted from the message.

   Click **Add** to launch the **Add Data Location** dialog box. You can then type in the location (for example, `$LocalEnvironment/File/Name`) or click **Edit** to launch the XPath Expression Builder.

   You can extract one or more fields from the message data and include them with the event. The fields can be simple or complex. Simple content is contained in the `wmb:applicationData/wmb:simpleContent` field of the event; complex data is contained in the `wmb:applicationData/wmb:complexContent` field.

   This facility is used commonly for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields. You can then use another application to provide an audit trail or to resubmit failed messages.

   For more information about configuring monitoring events that contain complex content, see “Including complex content in the event payload of a monitoring event ” on page 2802.
8. Optional: Select **Include bitstream data in payload** if the event must capture message bitstream data.

   If you select this option, you must provide the following information:

   Content
   Select from `Headers`, `Body`, and `All`.

   Encoding
   Select from `CData` (the original text, without encoding), `HexBinary`, and `base64Binary`.

   When this option is selected, it is possible that the bitstream that is included is not the same as the message that is written from a transport output node. Monitoring serializes the whole message tree, whereas transport output nodes use the parts of the message tree that they recognize. The bitstream that is included in a monitoring message is generated from all the headers that occur before the first message body in the message tree. Any headers that occur after the message body are not included.

9. Optional: Select the **Correlation** tab to provide details for event correlation.

   Every monitoring event must contain at least one correlation attribute, and can contain up to three. If you do not specify any correlation information, the first event source in the message flow allocates a unique identifier that all later event sources in the same transaction use.

   a) In the **Local transaction correlator** field, select one of the following options.

      Automatic
      The local correlator that is used by the most recent event for this invocation of the message flow is used. If no local correlator exists, a new unique value is generated.

      Specify location of correlator
      Enter a value, or click **Edit** to launch the XPath Expression Builder. The local correlator is read from the specified location in the message tree. Ensure that the specified location contains a correlator value that is unique to this invocation of the message flow.

   b) In the **Parent transaction correlator** field, select one of the following options to extract a correlation field from the parent transaction.

      Automatic
      The parent correlator that is used by the most recent event for this invocation of the message flow is used. If no parent correlator exists, no parent correlator is used.

      Specify location of correlator
      Enter a value, or click **Edit** to launch the XPath Expression Builder. The parent correlator is read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the parent correlator.

   c) In the **Global transaction correlator** field, select one of the following options to extract a correlation field from a global transaction.

      Automatic
      The global correlator that is used by the most recent event for this invocation of the message flow is used. If no global correlator exists, no global correlator is used.

      Specify location of correlator
      Enter a value, or click **Edit** to launch the XPath Expression Builder. The global correlator is read from the specified location in the message tree. Ensure that the specified location contains a suitable value for the global correlator.

10. Optional: On the **Transaction** tab, choose how the emission of monitoring events by a message flow is coordinated. You can choose to coordinate with the message flow transaction, in an independent unit of work, or not in a unit of work.

    Select one of the following options.

    Message flow
    The event, and all other events with this setting, are emitted only if the message flow commits its unit of work successfully.

    If the transaction start event is included in the message flow unit of work, but the message processing fails and this unit of work is not published, the transaction start event is included in an independent unit of work. This behavior ensures that your monitoring application receives a pair of events (start and rollback), rather than receiving a rollback event in isolation.

    Independent
    The event is emitted in a second unit of work, independent of the main unit of work. The event, and all other events with this setting, are emitted regardless of whether the main unit of work commits successfully.

    An independent transaction can be started only if the main transaction is either committed or rolled back. If the **`Commit count`** property of the flow is greater than one, or the **`Commit by`**

message group property is set, the events that are targeted for the independent transaction are emitted out of sync point. A message is also issued with this information.

None
The event is emitted out of sync point (not in any unit of work.) The event is emitted when the message passes through the event source, and is available for reading immediately.

Not all of these options are available on all event types. The default and allowed values are shown in the following table:

Event source	Allowed values	Default
Transaction start	Message flow Independent None	Message flow
Transaction end	Message flow None	Message flow
Transaction rollback	Independent None	Independent
Terminal	Message flow Independent None	Message flow

11. Click **Finish**.

    The **Events** table in the Monitoring tab of the Properties view for the node is updated with details of the event that you added, and the event is enabled.
12. Save the message flow.
13. When all events are added to the flow, add the message flow to the BAR file, and deploy the BAR file.

    Monitoring is inactive for the flow; deploying the BAR file does not activate monitoring.

What to do next

The monitoring properties for a node show all monitoring events that are defined for that node. You can edit the monitoring properties of a node to do the following tasks:

- Enable or disable a monitoring event.
- Add, delete, or change the monitoring events for the node.

Configuring monitoring event sources by using a monitoring profile

You can create a monitoring profile to configure your message flows to emit monitoring events.

Before you begin

Read the following topics:

- "Monitoring basics" on page 2772
- "Configuring monitoring for message flows" on page 2788

You must have a message flow that contains a node to which you want to add a monitoring event.

You can use XPath 1.0 expressions to configure a monitoring event.

About this task

To configure monitoring events by using a monitoring profile, complete the following tasks.

1. “Creating a monitoring profile” on page 2793
2. “Applying and activating a monitoring profile” on page 2798

Creating a monitoring profile

A monitoring profile XML file lists the event sources in the message flow that emits events, and defines the properties of each event.

Before you begin

A monitoring profile is an XML document that must be saved to a policy project. If you have a suitable policy project already, you can create a new XML file to save in that project. Otherwise, create a new policy project as described in “Creating policies with the IBM App Connect Enterprise Toolkit” on page 327.

About this task

A monitoring profile is an XML document that specifies the event sources in a message flow that emits events, and the properties of those events. The monitoring profile XML must conform to the XML schema file `MonitoringProfile.xsd`, which you can find in the samples directory of your IBM App Connect Enterprise installation:

```
Install_root/server/sample/Monitoring/MonitoringProfile.xsd
```

The XML file must have an extension of `.monprofile.xml`; for example, `monitoringProfileName.monprofile.xml`.

You can use either of the following methods to create a monitoring profile:

- Extract a monitoring profile from an existing message flow that is configured to use monitoring events. If you have a deployed message flow that was configured through the IBM App Connect Enterprise Toolkit Message Flow editor to use monitoring events, you can use either the command or the administration REST API to extract those monitoring properties from the deployed flow and create the equivalent monitoring profile (`.monprofile.xml` file) for the message flow. You can then use this profile as a starting point for other monitoring profiles, and edit them as required.

  For example, to extract a monitoring profile from a flow in an application that is running on an independent integration server, you can run a command similar to the following example:

  ```
  mqsireportflowmonitoring-i localhost -p 7600 -k <Application_name> -f <Flow_name>  -x <Path to
   your Toolkit Policy Project Folder>/<filename>.monprofile.xml
  ```

  After you successfully run the **mqsireportflowmonitoring** command, you must right-click on the Toolkit Policy Project in the IBM App Connect Enterprise Toolkit, and select **refresh**. The file appears in the **Navigator** view, in the **Other Resource** section. Alternatively, run the **mqsireportflowmonitoring** with the **-x** parameter set to the full path name of a file in a temporary location. Then use the Eclipse menu options **File**, and then **Import** to add `<filename>.monprofile.xml` into the Policy Project.
- Manually create a monitoring profile XML file (`.monprofile.xml`) from scratch, by completing the following steps.

Procedure

The following steps describe how to manually create a monitoring profile XML. Follow these steps for each `p:eventSource` element:

1. Use a text editor to create the monitoring profile XML file.

   For example, in the Toolkit right-click your policy project and then select **New** > **Other** > **XML / XML File**. Give the XML file a name with an extension of `.monprofile.xml` (for example,

monitoringProfileName.monprofile.xml). You can then use the Source view of the XML file to edit the monitoring profile.

The following example of a monitoring profile XML document contains a single event source to illustrate the structure.

```
<p:monitoringProfile
xmlns:p="http://www.ibm.com/xmlns/monitoring/11/profile" p:version="2.0">
    <p:eventSource p:enabled="true" p:eventSourceAddress="SOAPInput.transaction.Start">
        <p:eventPointDataQuery>
            <p:eventIdentity>
                <p:eventName p:literal="" p:queryText=""/>
            </p:eventIdentity>
            <p:eventCorrelation>
                <p:localTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:parentTransactionId p:queryText="" p:sourceOfId="automatic"/>
                <p:globalTransactionId p:queryText="" p:sourceOfId="automatic"/>
            </p:eventCorrelation>
           <p:eventFilter p:queryText="true()"/>
           <p:eventUOW p:unitOfWork="messageFlow" />
        </p:eventPointDataQuery>
        <p:applicationDataQuery>
            <p:simpleContent p:dataType="boolean" p:name="" p:targetNamespace="">
                <p:valueQuery p:queryText=""/>
            </p:simpleContent>
            <p:complexContent p:name="">
                <p:payloadQuery p:queryText=""/>
            </p:complexContent>
        </p:applicationDataQuery>
        <p:bitstreamDataQuery p:bitstreamContent="all" p:encoding="base64Binary"/>
    </p:eventSource>
</p:monitoringProfile>
```

The root element is p:monitoringProfile. It contains one or more p:eventSource elements, each of which specifies an event source and defines its properties. Each p:eventSource element contains:

- A p:eventPointDataQuery element that provides key information about the event.
- Optional: A p:applicationDataQuery element if the event payload includes data fields that are extracted from a message.
- Optional: A p:bitstreamDataQuery element if the event payload includes bitstream data from a message.

2. Specify the p:eventSource/@p:eventSourceAddress attribute.

   This string uniquely identifies the event source in the message flow. The string must follow the fixed format for transaction events and terminal events, as shown in the following table:

Event type	Event source address
Transaction Start event	*nodelabel*.transaction.Start
Transaction End event	*nodelabel*.transaction.End
Transaction Rollback event	*nodelabel*.transaction.Rollback
Terminal event	*nodelabel*.terminal.<Terminal>
All transaction events	transaction.all

The *nodelabel* attribute is the label of the node as known by the IBM App Connect Enterprise runtime components. The label also indicates whether the node is in a subflow. For example, flow A contains an instance of flow B as a subflow that is labeled myB. Flow B contains an instance of a Compute node that is labeled myCompute. The *nodelabel* for the Compute node is myB.myCompute.

In the emitted event, the address string of the event source is set in the wmb:eventData/@wmb:eventSourceAddress attribute.

The event type *All transaction Events* means that all nodes that can emit transaction events do emit start events, end events, or rollback events.

3. Optional: In the p:eventPointDataQuery/p:eventIdentity/p:eventName element, specify the name by which events from this event source are known.

   - If the event name is a fixed string, complete the p:eventName/@p:literal attribute.
   - If the event name is extracted from a field in the message, complete the p:eventName/@p:queryText attribute by specifying an XPath query.

   In the emitted event, the event name is set in the wmb:eventPointData/wmb:eventIdentity/@wmb:eventName attribute.

   If the p:eventName element is not supplied, @wmb:eventName in the emitted event defaults to @p:eventSourceAddress.
4. Optional: Complete the p:eventPointDataQuery/p:eventFilter/@p:queryText attribute by specifying an XPath expression to control whether the event is emitted. The expression must evaluate to true (the event is emitted), or false (the event is not emitted). The expression can reference fields in the message tree, or elsewhere in the message assembly. If an event does not contain an eventFilter element, the event is always emitted.

   By using this facility, you can tailor event emissions to your business requirements, by filtering out events that do not match a set of rules. This facility can reduce the number of events that are emitted, and reduce the workload on your monitoring application.
5. Optional: If the event is to contain selected data fields that are extracted from the message, complete the p:applicationDataQuery element.

   You can extract one or more fields from the message data and include it with the event. The fields can be simple or complex.

   - For each simple data field, complete a p:simpleContent element:
     - Complete the p:simpleContent/p:valueQuery/@p:queryText attribute by specifying an XPath query.
     - Complete the p:simpleContent/@p:name, @p:namespace, and @p:dataType attributes. The @p:dataType value must be one of boolean, date, dateTime, decimal, duration, integer, string, or time.
   - For each complex data field, complete a p:complexContent element:
     - Complete the p:complexContent/p:payloadQuery/@p:queryText attribute by specifying an XPath query.
     - Complete the p:complexContent/@p:name attribute.

   This facility is commonly used for communicating significant business data in a business event. If the event contains the input bit stream, this facility can also be used to extract key fields. You can then use another application to provide an audit trail or to resubmit failed messages.

   In the emitted event, the extracted data is set in the wmb:applicationData/wmb:simpleContent and wmb:applicationData/wmb:complexContent elements.
6. Optional: If the event is to capture message bitstream data, complete the p:bitstreamDataQuery element.

   - Complete the @p:bitstreamContent attribute. The attribute value must be one of headers, body, or all.
   - Complete the @p:encoding attribute. The attribute value must be one of CDATA, base64Binary, or hexBinary.

   In the emitted event, the extracted bitstream data is set in the wmb:bitstreamData/wmb:bitstream element.

7. Optional: Complete the p:eventPointDataQuery/p:eventCorrelation element.

   Every emitted monitoring event can contain up to three correlation attributes. If no correlation information is specified in the monitoring profile, no correlation attributes are used.

   a. Complete the p:localTransactionId element.

      - To reuse the local correlator from the environment tree, set the p:localTransactionId/@p:sourceOfId attribute to automatic. If no local correlator exists yet, a new unique value is generated and saved in the environment tree.
      - To use a value that is contained in a location in the message, set the p:localTransactionId/@p:sourceOfId attribute to query, then complete the p:localTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a correlator value that is unique to this invocation of the message flow. The value is saved in the environment tree.

   b. Complete the p:parentTransactionId element.

      - To reuse the parent correlator from the environment tree, set the p:parentTransactionId/@p:sourceOfId attribute to automatic. If no parent correlator exists, no parent correlator is used.
      - To use a value that is contained in a location in the message, set the p:parentTransactionId/@p:sourceOfId attribute to query, then complete the p:parentTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the parent correlator. The value is saved in the environment tree.

   c. Complete the p:globalTransactionId element.

      - To reuse the global correlator from the environment tree, set the p:globalTransactionId/@p:sourceOfId attribute to automatic. If no global correlator exists, no global correlator is used.
      - To use a value that is contained in a location in the message, set the p:globalTransactionId/@p:sourceOfId attribute to query, then complete the p:globalTransactionId/@p:queryText attribute by specifying an XPath query. Ensure that the specified location contains a suitable value for the global correlator. The value is saved in the environment tree.

8. Complete the p:eventPointDataQuery/p:eventUOW element. This element determines how the emission of monitoring events by a message flow is coordinated. You can choose to coordinate with the message flow transaction, in an independent unit of work, or not in a unit of work.

   Set the p:eventUOW/@p:unitOfWork attribute to one of the following values:

   Message flow
   The event, and all other events with this setting, are emitted only if the message flow commits its unit of work successfully.

   If the transaction start event is included in the message flow unit of work, but message processing fails and this unit of work is not published, the transaction start event is included in an independent unit of work. This behavior ensures that your monitoring application receives a pair of events (start and rollback), rather than receiving a rollback event in isolation.

   Independent
   The event is emitted in a second unit of work, independent of the main unit of work. The event, and all other events with this setting, are emitted regardless of whether the main unit of work commits successfully.

   An independent transaction can be started only if the main transaction is either committed or rolled back. If the **Commit count** property of the flow is greater than one, or the **Commit by**

message group property is set, the events that are targeted for the independent transaction are emitted out of sync point. A message is also output with this information.

None
The event is emitted out of sync point (not in any unit of work.) The event is emitted when the message passes through the event source, and is available for reading immediately.

Not all of these options are available on all event types. The allowed values are shown in the following table.

Event Type	Allowed values
transaction.Start	messageFlow independent none
transaction.End	messageFlow none
transaction.Rollback	independent none
terminal	messageFlow independent none

If you do not include the `eventUOW` element for an event source, transactionality for all events that are output from that source (except transaction.rollback events) defaults to `messageFlow`. Transactionality for transaction.rollback events defaults to `independent`.

9. If you use XPath to select the value of a field, and if the XPath query contains a component that has an XML namespace, the XPath will contain a namespace prefix for the namespace. The namespace prefix in all `prefixMapping` elements in a monitoring profile must be unique.
10. Ensure that the monitoring profile XML conforms to the XML schema file `MonitoringProfile.xsd`, which you can find in the samples directory of the IBM App Connect Enterprise installation (*Install_root*`/server/sample/Monitoring/MonitoringProfile.xsd`).
11. Save the XML file with an extension of `.monprofile.xml` (for example, *monitoringProfileName*`.monprofile.xml`) to your policy project.

What to do next

Deploy the monitoring profile, as described in Deploying and redeploying a monitoring profile.

Deploying and redeploying a monitoring profile

Deploy or redeploy a monitoring profile in a Policy Project.

Before you begin

Create a monitoring profile and save it to a Policy Project by following the instructions in "Creating a monitoring profile" on page 2793.

About this task

If you deploy a monitoring profile, it must be deployed in a policy project, and the XML file must have an extension of `.monprofile.xml`. To deploy a monitoring profile, complete step 1 and step 2 in the Procedure.

You can redeploy a monitoring profile by completing step 3 in the Procedure.

You can override a deployed monitoring profile by completing step 4 in the Procedure.

As an alternative to deploying the monitoring profile, you can place the `monprofile.xml` file in the integration server's overrides directory, as described in step 5 in the Procedure.

Procedure

1. Create a BAR file to contain your Policy Project by completing the following steps:
   a. In the IBM App Connect Enterprise Toolkit, switch to the Integration Development perspective.
   b. In the Application Development view, right-click the BAR file that contains the message flows, then click **Open with** > **BAR Editor**.
   c. Click the **Manage** tab.
   d. Click the message flow on which you want to set the monitoring profile configurable service. The properties that you can configure for the message flow are displayed in the **Properties** view.
   e. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.
   f. Save the BAR file.
2. Deploy the BAR file to your integration server.
3. Optional: If you make further changes to the monitoring profile, you can save it to the Policy Project, edit the BAR file, and redeploy it to your integration server. All message flows that use the monitoring profile are stopped and restarted to use the new values.

   For more information, see Monitoring profiles.
4. Optional: You can also override a deployed monitoring profile by putting an updated `.monprofile.xml` file in the integration server's `overrides` directory, as described in "Order of precedence for overrides" on page 326.

   You must restart the integration server to pick up any newly added or changed `monprofile.xml` file in the integration server's `overrides` directory.

As an alternative to deploying the monitoring profile, you can place the `monprofile.xml` file in the integration server's overrides directory, by completing the following step:

5. Optional: As an alternative to deploying the monitoring profile, you can place the `monprofile.xml` file in the integration server's `overrides` directory.

   For integration servers that are managed by an integration node, the location of the `overrides` directory is `$MQSI_WORKPATH/components/<Node name>/overrides/node.conf.yaml`.

   For independent integration servers, the location of the `overrides` directory is `<work directory>/overrides`.

   You must restart the integration server to pick up any newly added or changed `monprofile.xml` file in the integration server's `overrides` directory.

What to do next

Monitoring for the flow is inactive. Deploying the BAR file does not activate monitoring for the flow. After you create and deploy the monitoring profile in a policy project, you must apply and activate the monitoring profile by using either the command or the administration REST API. For more information, see Applying and activating a monitoring profile.

Alternatively, you can specify the monitoring profile as a BAR file override as described in Configurable properties in a file.

Applying and activating a monitoring profile

After you create and deploy a monitoring profile, apply it to one or more message flows and then activate monitoring for the flows.

Before you begin

Create a monitoring profile by following the instructions in "Creating a monitoring profile" on page 2793.

Deploy or redeploy a monitoring profile by following the instructions in "Deploying and redeploying a monitoring profile" on page 2797.

About this task

Applying means associating a deployed monitoring profile with a message flow. Applying can be done statically or dynamically.

Activating means enabling monitoring. Activating can be done statically or dynamically.

Apply and activate your monitoring profile by completing the following steps:

Procedure

1. Apply the monitoring profile to one or more message flows by completing one of the following steps:

   Statically. Before you deploy the message flow, use the BAR File editor to apply a monitoring profile to one or more message flows. Set the message flow property **`Monitoring Profile Name`**, and then deploy the BAR file by completing the following steps:

   a. In the IBM App Connect Enterprise Toolkit, switch to the Integration Development perspective.
   b. In the Application Development view, right-click the BAR file that contains the message flows, then click **Open with** > **BAR Editor**.
   c. Click the **Manage** tab.
   d. Click the message flow on which you want to set the monitoring profile configurable service. The properties that you can configure for the message flow are displayed in the **Properties** view.
   e. In the **Monitoring Profile Name** field, enter the name of a monitoring profile.
   f. Save the BAR file.
   g. Deploy the BAR file. The monitoring profile must be deployed in the same BAR file as the message flow.

   You can also apply a default monitoring profile for any message flow in an integration server, by setting properties in the *Defaults* section of the `server.conf.yaml` file. For example:

```
Defaults:
```

```
            monitoringProfile: 'myIS1MonitoringProfile'       # Default Monitoring profile
```

A default monitoring profile is applied only to message flows that do not have either a configured monitoring profile, or any toolkit monitoring events defined on it.

Dynamically. You can choose from two alternative methods to apply the monitoring profile. You can run the **mqsichangeflowmonitoring** command, or use the IBM App Connect Enterprise administration REST API with the `curl` command:

- Run the **mqsichangeflowmonitoring** command with the **--monitoring-profile** parameter. For example, to apply monitoring to a single message flow *messageflow1* in application *Application1* in integration server *IS1* of the integration node *myBroker*, run the following command:

```
mqsichangeflowmonitoring myBroker -e IS1 -k Application1 -f messageflow1 -c active --
monitoring-profile profileName
```

Where *profileName* is either *{policyProject}:profileName* or *profileName*. If you do not include the prefix *{policyProject}:*, then you must deploy the *profileName* in the configured default policy project.

For more information about the **mqsichangeflowmonitoring** command, see command.

- Use the REST API to associate the monitoring profile with a message flow. Use a POST to make a change that applies until the message flow is restarted or redeployed. For example, for a message flow deployed in a managed integration server:

```
curl -v -X POST "http://<host>:<port>/apiv2/servers/<server name>/applications/<app
 name>/messageflows/<flow name>/attach-monitoring-profile?project=<policy project
 name>&profile=<profile-name>
```

Use a PATCH to have the change persisted though restart or deploy. For example:

```
curl -v -X PATCH -H "Content-Type: application/json" -d Monitoring\monProf.json "http://
<host>:<port>/apiv2/servers/<server name>/applications/<app name>/messageflows/<flow name>
```

2. Activate the monitoring profile by completing one of the following steps:

Statically.

- You can activate monitoring, and optionally set the event format, for all applications and message flows in an integration server, by setting properties in the *Monitoring.MessageFlow* section of the `server.conf.yaml` file. For example:

```
Monitoring:
           MessageFlow:
           publicationOn: 'active'         # choose 1 of : active|inactive, default
 inactive
                                           # Ensure Events.BusinessEvents.MQ|MQTT is set
```

```
            #eventFormat: 'MonitoringEventV2'  # choose 1 of : MonitoringEventV2|WMB
```

This integration server level setting is used by applications and message flows only if they are left with the default setting of *inherit*. If you use the following dynamic option to set monitoring to *active* or *inactive*, it overrides the integration server level setting.

Dynamically. You can choose from two alternative methods to activate the monitoring profile. You can run the **mqsichangeflowmonitoring** command, or use the IBM App Connect Enterprise administration REST API with the `curl` command:

- Use the **mqsichangeflowmonitoring** command. For example, to activate monitoring for a single message flow *messageflow1* in application *Application1* in integration server *IS1* of the integration node *myBroker*, run the following command:

```
mqsichangeflowmonitoring myBroker -e IS1 -k Application1 -f messageflow1 -c active --
monitoring-profile profileName
```

Where *profileName* is either *{policyProject}:profileName* or *profileName*. If you do not include the prefix *{policyProject}:*, then you must deploy the *profileName* in the configured default policy project.

- Use the IBM App Connect Enterprise administration REST API to activate the monitoring profile. For example, to activate monitoring on a single message flow `MonitoringAppFlow1` in the application `MonitoringApp` that is deployed to an independent integration server, run the following command:

```
curl -X POST http://hostname:port/apiv2/applications/MonitoringApp/messageflows/
MonitoringAppFlow1/start-monitoring
```

For more information about activating monitoring by using the IBM App Connect Enterprise administration REST API, see "Activating monitoring by using the administration REST API" on page 364.

Enabling and disabling event sources

When events are configured for a message flow and deployed to the integration node, you can enable and disable individual events. If you enable and disable events from the command line, you do not need to redeploy the message flow. If you use the Message Flow editor to enable or disable events, you must redeploy the flow.

About this task

To enable or disable events, follow the instructions in one of the following sections:

- "Enabling and disabling events from the command line" on page 2801
- "Enabling and disabling events from the Message Flow editor" on page 2802

Enabling and disabling events from the command line

Before you begin

Configure events by using monitoring properties in the Message Flow editor:

- "Configuring monitoring event sources by using monitoring properties" on page 2788

About this task

To enable or disable events from the command line, complete the following steps:

Procedure

Use the **mqsichangeflowmonitoring** command, as shown in the following examples.

- To enable monitoring for all message flows in library `lib1` in application `app1`, on the specified integration server:

```
mqsichangeflowmonitoring -c active -i localhost -p 7600 -k app1 -y lib1 -f "*"
```

- To disable monitoring for message flow `flow3` in application `app0`, which is on integration server `server0` on integration node `broker1`:

```
mqsichangeflowmonitoring -c inactive broker1 -e server0 -k app0 -f flow3
```

You can enable or disable multiple events at the same time. The change of state takes effect immediately.

If you configured events by using monitoring properties, the change persists if the message flow is restarted, but is lost if the message flow is redeployed. To make the change permanent, you must also update the monitoring properties.

Enabling and disabling events from the Message Flow editor

Before you begin

Configure events by using monitoring properties in the Message Flow editor (see "Configuring monitoring event sources by using monitoring properties" on page 2788).

About this task

To enable and disable event sources by using the Message Flow editor, complete the following steps.

Procedure

1. Open the message flow in the editor.
2. Click the canvas.
3. Select the **Monitoring** tab in the Properties view.

   The monitoring events that you defined are shown.
4. Select **Enabled** for each event that you want to enable.

   To disable a single event, clear the checkbox for that event. To disable all events, click **Uncheck All**.
5. Save the message flow.
6. Rebuild and redeploy the BAR file.

Including complex content in the event payload of a monitoring event

You can include parts of a message assembly in a monitoring event, by configuring an XPath expression in the event payload of a monitoring event.

Most events contain data that is taken from fields in the message tree or from elsewhere in the message assembly. You specify the **`Data location`** in the **`Event Payload`** section of the Monitoring Event editor.

If the configured XPath in the **`Data Location`** of the **`Event Payload`** resolves to a simple element (`"name:value"`), the monitoring event includes a `"simpleContent"` section. For example, with an XPath `"$Root/DFDL/CSV/header/head_field2"`, where `"head_field2"` is a simple string element, the monitoring event formatted in JSON could contain the following code:

```
"applicationData": {
"simpleContent": [{
"elementName": "head_field2",
"elementValue": "h2",
"elementDataType": "string"
}],
```

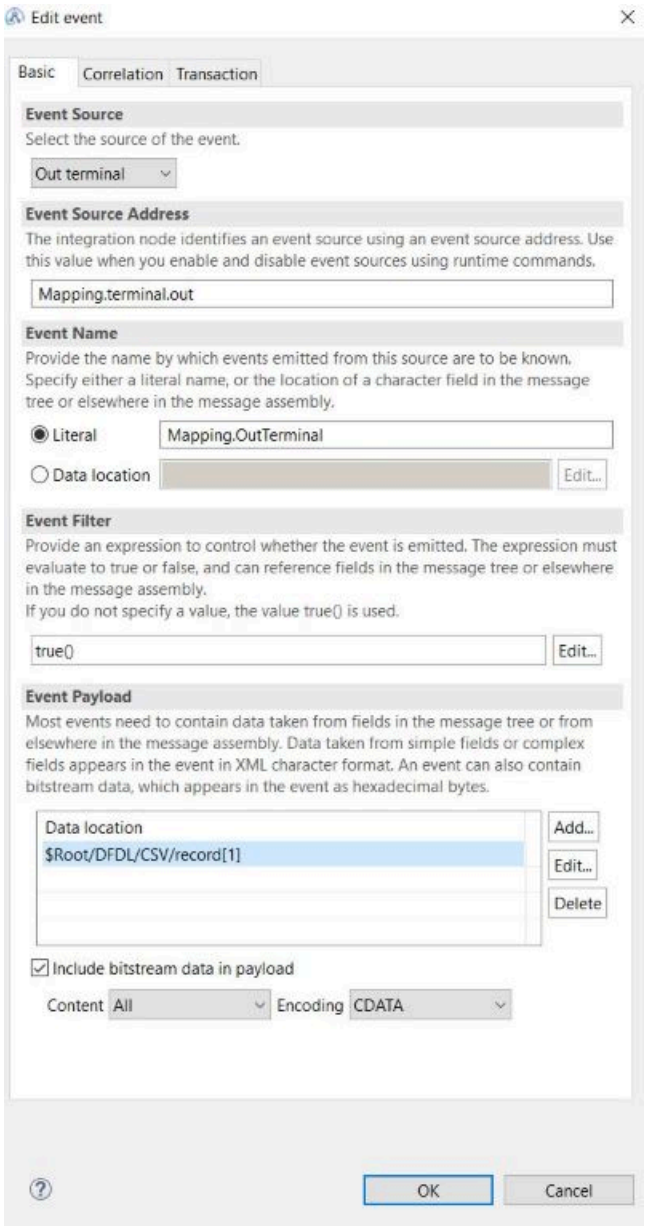


Figure 38. Monitoring Event editor

If the configured XPath in the **Data Location** of the **Event Payload** resolves to a complex element (a sub-tree), the monitoring event includes a `"complexContent"` section. For example, with an XPath `"$Root/DFDL/CSV/record[1]"`, where`"record"` is a complex element that has child elements, the monitoring event formatted in JSON could contain the following:

```
"applicationData": {
"complexContent": [{
"elementName": "record",
"elementValue": {
"bitstream": {
"field1": "Field1Val",
"field2": "Field2Val"
}
}
}]
```

If the configured XPath in the **Data Location** of the **Event Payload** resolves to a repeating simple or complex element, there is a `"simpleContent"` or `"complexContent"` section for each instance.

If an XPath query ends in a wildcard, each element node or attribute node appears in a separate `complexContent` or `simpleContent` folder in the emitted event.

If required, you can configure the **`Data Location`** of the **`Event Payload`** to resolve to a complex element in a message assembly that is in a message domain that does not match the format of the monitoring event. For example, the XPath in the **`Data Location`** could be `"$Root/XML/... "` and the monitoring event format could be JSON, or the XPath in the **`Data Location`** could be `"$Root/DFDL/... "` and the monitoring event format could be JSON or XML. However, if the message domain of the complex element and the format of the monitoring event are different, consider the following points:

- XML monitoring event format:
  - The names of elements resolved by the **`Event Payload Data Location`** XPath must be valid XML element names.
  - MRM XML attributes become elements in the monitoring event. The resolved element must not have an attribute and a child element with the same name.
- JSON monitoring event format:
  - The names of elements resolved by the **`Event Payload Data Location`** XPath must be valid JSON key fields.
  - XMLNSC, XMLNS, and MRM attributes become JSON fields. The resolved element must not have an attribute and a child element with the same name.
  - If the resolved element has a child element that repeats, the JSON monitoring event will have an object with repeating keys, which might lead to a "duplicate key" error.

Activating monitoring

Use the **`mqsichangeflowmonitoring`** command to activate monitoring after you configure monitoring event sources.

Before you begin

Configure monitoring event sources by using either monitoring properties or a monitoring profile:

- “Configuring monitoring event sources by using monitoring properties” on page 2788
- “Configuring monitoring event sources by using a monitoring profile” on page 236

About this task

If monitoring is activated for a message flow, and monitoring properties or a monitoring profile are not configured for the flow, transaction events are emitted. You must ensure that event publication for monitoring messages is enabled and is correctly configured, including specifying whether events are emitted in XML, JSON, or both. For more information, see “Configuring the publication of event messages” on page 2807.

You can start or stop monitoring and specify the appropriate event format by setting properties in the configuration file for your integration server (`server.conf.yaml`.

```
Monitoring:
  MessageFlow:
    #publicationOn: inactive           # Choose 1 of : active|inactive, default inactive
                                       # Ensure Events.BusinessEvents.MQ|MQTT is set
    #eventFormat: MonitoringEventV2    # When BusinessEvents output format is 'xml', set
 'MonitoringEventV2' or 'WMB'. Default 'MonitoringEventV2'.
```

Changes that you make to the properties in the `.conf.yaml` file take effect when the integration server is restarted, and apply to every flow in the integration server.

When you configure monitoring for your flows, you can use the command to activate or deactivate monitoring without restarting the integration server. The options that you set with this command remain in

effect until you run the command again or restart the integration server. When you restart the integration server, the settings that are specified in the `server.conf.yaml` file take effect.

To activate monitoring, specify the **`-c active`** parameter on the **mqsichangeflowmonitoring** command. To deactivate monitoring, specify **`-c inactive`**. You can activate or deactivate monitoring for all message flows in all integration servers. Alternatively, you can specify the integration servers, applications, libraries, or message flows for which monitoring is to be turned on or off.

For more information, see the command.

You can also use a monitoring profile to configure your message flows to emit monitoring events. When you create a monitoring profile, you deploy it to a policy project and then activate it on a specific message flow, application, or integration server. Activate the monitoring profile by using either the **mqsichangeflowmonitoring** command or the administration REST API. Alternatively, you can specify a monitoring profile as a BAR file override, as described in Configurable properties in a file. You can override a deployed monitoring profile by putting an updated `.monprofile.xml` file in the integration server's overrides directory, as described in “Order of precedence for overrides” on page 326.

You can check the monitoring options that are set for a message flow, application, or library, by using the **mqsireportflowmonitoring** command. For more information, see command.

Procedure

Configure monitoring options for a message flow by setting properties in the Monitoring section of the configuration file for your integration node or server (`node.conf.yaml` or `server.conf.yaml`):

1. Open the appropriate `node.conf.yaml` or `server.conf.yaml` configuration file by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are invalid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. To configure message flow monitoring options, set the following properties in the configuration file:
   a) Enable message flow monitoring by setting the **publicationOn** property to `active`. By default, message flow monitoring is turned off.
   b) Specify the correct format of monitoring events by setting the **eventFormat** property.
      Valid options are `MonitoringEventV2` and `WMB`.

   `MonitoringEventV2` is the default format that is used by default in IBM App Connect Enterprise 12.0. It provides some additional data that relates to Shared Libraries and the CCSID of encoded message bodies.

   Specify `WMB` if you want the monitoring events to be emitted in the same format as is used in IBM Integration Bus 10.0.

   For more information, see “The monitoring event” on page 2778.
3. Save the file and restart the integration server for the changes to take effect.

   Changes that you make to the properties in `.conf.yaml` files take effect when the integration server is restarted. For more information about how to start an integration server, see “Starting an integration server” on page 250.

When monitoring is configured, you can use the command to dynamically start or stop message flow monitoring, without restarting the integration server:

4. Optional: To activate monitoring, use the **-c active** parameter. You can activate monitoring for all message flows in all integration servers, or you can specify the integration servers, applications, libraries, message flows for which monitoring is to be activated.
   For example,

   To turn on monitoring for all message flows in library `lib1` in application `app1`, on the specified integration server, use the following command:

   ```
   mqsichangeflowmonitoring -c active -i localhost -p 7600 -k app1 -y lib1 -f "*"
   ```

   By default, the settings that you specify with this command persist when the integration server or node is restarted and when the message flow is redeployed. If you do not want the settings to persist, you can specify the **--non-persist** parameter. The **--non-persist** parameter causes the settings to remain in effect only until a restart or redeploy, at which point the settings that are specified in the `server.conf.yaml` or `node.conf.yaml` configuration file take effect.
5. Optional: To deactivate monitoring, use the **-c inactive** parameter. You can deactivate monitoring for all message flows in all integration servers, or you can specify the integration servers, applications, libraries, message flows for which monitoring is to be turned off.
   For example,

   To turn off monitoring for message flow `flow3` in application `app0`, which is on integration server `server0` on integration node `broker1`, use the following command:

   ```
   mqsichangeflowmonitoring -c inactive broker1 -e server0 -k app0 -f flow3
   ```

Collecting monitoring messages for all message flows in a server

You can collect monitoring messages for all message flows in a server by deploying a monitoring profile in a policy project, and configuring the `server.conf.yaml` file.

Before you begin

Create a monitoring profile that uses the **Event type** *All transaction events*, and the **Event source address** *transaction.all* as described in “Creating a monitoring profile” on page 2793.

About this task

You can collect monitoring messages for all message flows in a server by creating a monitoring profile that is configured for all transaction events. You must then add the profile to a policy project before you deploy it to the server. You must then configure the `server.conf.yaml` file for all transactions, enable the collection of the events, and enable the writing of the events to a file.

To collect monitoring messages for all message flows in a server, complete the following tasks.

Procedure

1. If you do not have a monitoring profile that uses the **Event type** *All transaction events*, and the **Event source address** *transaction.all*, create one as described in “Creating a monitoring profile” on page 2793.
2. Add the monitoring profile to a policy project and deploy it to the integration server. For more information, see “Deploying and redeploying a monitoring profile” on page 2797.
3. Configure the default monitoring profile for all flows in the `server.conf.yaml` file. In the following example, the monitoring profile is called *AllTransactions_Test* and is in a policy project called *PolProj* :

   ```
   Defaults:
     Policies:
       monitoringProfile: '{PolProj}:AllTransactions_Test'       # Default Monitoring profile
   ```

4. Configure the `server.conf.yaml` file to enable the collection of monitoring events:

```
Monitoring:
  MessageFlow:
    publicationOn: 'active'         # choose 1 of : active|inactive, default inactive
                                       # Ensure Events.BusinessEvents.MQ|MQTT is set
    #eventFormat: 'MonitoringEventV2'  # choose 1 of : MonitoringEventV2|WMB
```

5. Configure the `server.conf.yaml` file to enable the writing to file of monitoring events:

```
Events:
  BusinessEvents: # Monitoring events
    File:
      enabled: true            # Set true or false, default false
      #outputFormat: 'json'      # Set comma separated list of one or more of : json,xml.
 Defaults to 'json'
```

The publishing of the events can be done over WebSphere MQ, MQTT, or ELK.

Configuring and subscribing to performance and monitoring events

Integration servers use a pub/sub broker to publish event messages (as fixed topics) in response to changes in configuration, state, or operational status. You can then monitor these events by subscribing to the topics.

Operational events are required by the message flow statistics and resource statistics features of IBM App Connect Enterprise. If you want to collect and view statistics, you must ensure that event publication is enabled and that a pub/sub broker has been configured. For more information, see the following topics:

- “Configuring the publication of event messages” on page 2807
- “Configuring the built-in MQTT pub/sub broker” on page 2811

To monitor the events, you subscribe to the appropriate topics. For more information, see “Subscribing to event message topics” on page 2813.

For detailed information on the architecture of the event messages, see “IBM App Connect Enterprise event messages” on page 2816.

Configuring the publication of event messages

You can configure the publication of event messages to specify the pub/sub broker to which the messages are published, whether messages are published for an event message group, and whether events are emitted in XML, JSON, or both.

Before you begin

Read the following topics:

- “Publish/subscribe overview” on page 1236
- “Configuring and subscribing to performance and monitoring events” on page 2807

About this task

- You need to enable your required publication of events as follows:

  For Operational Events or Statistics and Accounting, use the **mqsichangeflowstats** command or set Statistics.Snapshot.publicationOn or Statistics.Archive.archivalOn in the `.conf.yaml` file.

  For Business Events or Message Flow Monitoring, use the **mqsichangeflowmonitoring** command, or set Monitoring.MessageFlow.publicationOn to `active` in the `.conf.yaml` file.

  Integration node and independent integration server event messages can be published to the following destinations:

  – The default IBM MQ queue manager of an integration node or independent integration server
  – The integration node MQTT pub/sub broker (not available for independent integration servers)
  – An external IBM MQ queue manager, by configuring an MQEndpoint policy
  – An external MQTT pub/sub broker, by configuring an MQTTPublish policy
  – The file system, in rotatable log files
  – A Logstash input plug-in in an Elasticsearch, Logstash, and Kibana (ELK) stack

  You can configure both IBM MQ and MQTT pub/sub brokers.

  The default pub/sub broker that is used for each group of event messages (`OperationalEvents` and `BusinessEvents`) depends on your IBM App Connect Enterprise deployment:

  IBM MQ is not installed, or IBM MQ is installed but a queue manager is not specified on the integration node.
  By default, the messages are published to the following locations:

  – `OperationalEvents` messages are published to the built-in MQTT pub/sub broker.
  – `BusinessEvents` messages are not published.

  IBM MQ is installed and a default queue manager is specified on the integration node or independent integration server.
  By default, the messages are published to the following locations:

  – `OperationalEvents` messages are published to the IBM MQ queue manager, and for the integration node only, to the built-in MQTT pub/sub broker.
  – `BusinessEvents` messages are published to the IBM MQ pub/sub broker.

  In either configuration, if you want to publish `BusinessEvents` messages to the built-in MQTT pub/sub broker, you must explicitly enable the publication of the `BusinessEvents` group to the built-in MQTT pub/sub broker.

  You can change the settings for the publication of `OperationalEvents` and `BusinessEvents`, and the integration node built-in MQTT broker, either by using the **mqsichangeproperties** command or by editing the `node.conf.yaml` or `server.conf.yaml` configuration file. You can configure specific IBM MQ or MQTT connection policies, for each of the `OperationalEvents` and `BusinessEvents` event message groups.

  If you want to use an external MQTT server instead of the built-in MQTT broker, you can create a policy that contains the connection details that you want to use. For more information about creating policies, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.

  If you want to use the IBM MQ pub/sub broker and you want to use a different queue manager to the queue manager that is specified on the integration server, you can create a policy that contains the connection details that you want to use. For more information about creating policies, see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327.

  To use a secured IBM MQ pub/sub broker or an MQTT server that requires a username and password, use the **mqsisetdbparms** command to define the credentials to use for the connection. If a specific identity is not defined in the configured policy, the integration node uses the security identity **pubsubDefault** when event messages are published. If **pubsubDefault** is associated with security

credentials by using the **mqsisetdbparms** command, then these credentials are used; otherwise, no credentials are used. For more information, see command.

For more information about of all the event messages that can be published, see "Subscribing to event message topics" on page 2813.

You can configure whether events are published, and where they are published to, either by using the **mqsichangeproperties** command, or by setting properties in the `server.conf.yaml` or `node.conf.yaml` configuration file. Configure the publication of events by completing the steps in one of the following tasks:

- "Using the mqsichangeproperties command" on page 2809
- "Modifying the server.conf.yaml or node.conf.yaml file" on page 2810

Using the `mqsichangeproperties` command

About this task

Follow these steps to configure the publication of event messages by using the **mqsichangeproperties** command:

Procedure

1. Use the **enabled** property of the **mqsichangeproperties** command to enable or disable the publication of event messages by the specified pub/sub broker.

   All integration node events are categorized by event message group (`OperationalEvents` and `BusinessEvents`). You can enable or disable the publication of event messages that are based on the event message group. For example, enter the following command to enable the publication of event messages to an MQTT broker for the publication of `BusinessEvents` messages:

   ```
   mqsichangeproperties INODE -b pubsub -o BusinessEvents/MQTT -n enabled -v true
   ```

2. If you want to use an external MQTT server for the publication of event messages, or if you want to use a specific IBM MQ queue manager, use the **policy** property of the **mqsichangeproperties** command. You can use the **policy** to specify the location of the policy that contains the connection details.
   For example:

   ```
   mqsichangeproperties INODE -b pubsub -o BusinessEvents/MQTT -n policy -v
    {DefaultPolicies}:example_mqtt_policy
   ```

3. Optional: If you want to publish the event messages to rotatable log files in the file system, complete the following steps:

   a) Enable output to a file by using the **mqsichangeproperties** command, as shown in the following example:

   ```
   mqsichangeproperties INODE -b pubsub -o BusinessEvents/File -n enabled -v true
   ```

   b) Set the output format of the event messages to either `json`, `xml`, or `json.xml` as shown in the following example:

   ```
   mqsichangeproperties INODE -b pubsub -o BusinessEvents/File -n outputFormat -v json
   ```

4. Optional: If you want to publish the event messages to a Logstash input plug-in in an Elasticsearch, Logstash, and Kibana (ELK) stack, complete the following steps:
   a) Enable output to a Logstash input plug-in in an Elasticsearch, Logstash, and Kibana (ELK) stack by using the **mqsichangeproperties** command, as shown in the following example:

```
mqsichangeproperties INODE -b pubsub -o BusinessEvents/ELK -n enabled -v true
```

   b) Set the ELK "elkConnections" to match a name of a ELKConnections section by using the **mqsichangeproperties** command, as shown in the following example:

```
mqsichangeproperties INODE  -b pubsub -o BusinessEvents/ELK -n elkConnections -v
 elkConnection1
```

   The referenced ELKConnections named section must be configured as described in "Reporting logs and monitoring events to a Logstash input in an ELK stack" on page 2869.
5. Ensure that the subscription is configured correctly so that the required event messages are received correctly following the configuration of the publication details.

   For example, if subscription was previously turned off in the web user interface, you must turn it on before published event messages can be received and processed for functions. For example, message flow statistics.

Modifying the server.conf.yaml or node.conf.yaml file

About this task

Follow these steps to configure the publication of event messages by setting properties in the `server.conf.yaml` or `node.conf.yaml` configuration file:

Procedure

1. Use a YAML editor to open the `.yaml` file.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, which are not accepted in YAML and would cause your configuration to fail. If you choose to use a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

   For more information about working with YAML, see http://www.yaml.org/start.html.
2. Set the appropriate property values in the Events section of the `.yaml` file:

```
Events:
  OperationalEvents: # Message flow and Resource statistics plus Workload management
    MQ:
      #policy: ''      # Specify a {policy project}:policy if not using
 'defaultQueueManager'
      #enabled: true   # Set true or false, default false
      #format: ''      # Set string or none
    MQTT:
      #policy: ''      # Specify a {policy project}:policy
      #enabled: false  # Set true or false, default false
      #format: ''      # Set string or none
  BusinessEvents:    # Business monitoring events
    MQ:
      #policy: ''      # Specify a {policy project}:policy if not using
 'defaultQueueManager'
      enabled: true    # Set true or false, default false
      #format: ''      # Set string or none
      #outputFormat: 'xml' # Set comma separated list of one or more of : json,xml. Defaults
 to 'xml'
    MQTT:
      #policy: ''      # Specify a {policy project}:policy
      #enabled: true   # Set true or false, default false
      #format: ''      # Set string or none
      #outputFormat: 'xml' # Set comma separated list of one or more of : json,xml. Defaults
 to 'xml'
```

```
    ELK:
      #enabled: false            # Set true or false, default false
      #outputFormat: 'json'      # Set json, default json
      #elkConnections: ''        # Name of the ELK connection to use, for example
 'elkConnection1', must be defined in the ELKConnections section below.
    File:
      #enabled: false       # Set true or false, default false
      #outputFormat: 'json' # Set comma separated list of one or more of : json,xml. Defaults
 to 'json'
```

The business transaction monitoring capability requires your monitoring events to be published as MQ publications in `xml` format. If you intend to use business transaction monitoring, either set the **outputFormat** property in the `Events.BusinessEvents.MQ` section of the `.yaml` file to include `'xml'`, or leave it commented out so that `xml` format is assigned by default.

The referenced ELKConnections named section must be configured as described in "Reporting logs and monitoring events to a Logstash input in an ELK stack" on page 2869.

3. Save the changes to the `.yaml` file.
4. Restart the integration node or integration server for the changes to take effect. The properties that you set in the `.yaml` file take effect when the integration node or server is started. If you modify these properties again, you must also start the integration node or server again for the later changes to be applied.

Configuring the built-in MQTT pub/sub broker

Configure the built-in MQTT pub/sub broker for an integration node, by using the **mqsichangeproperties** command, or by updating the configuration file in the *work_path*\components*integrationNodeName*\config directory.

Before you begin

Read the following topics:

- "Publish/subscribe overview" on page 1236
- "Configuring and subscribing to performance and monitoring events" on page 2807

About this task

The built-in MQTT broker is enabled by default. It is the default transport for the publication of operational and admin events by an integration node, unless:

- IBM MQ is installed.
- An IBM MQ queue manager is specified on the integration node.

For more information, see "Configuring the publication of event messages" on page 2807.

The built-in MQTT pub/sub broker is only applicable to an integration node. If you want to publish event messages for an independent integration server, you must configure an MQTTPublish policy to use an external MQTT pub/sub broker. For more information, see "Configuring and subscribing to performance and monitoring events" on page 2807.

You can modify the configuration of the built-in MQTT broker by using the **mqsichangeproperties** command. Use the **enabled** property for the MQTTServer object in the pub/sub component to enable or disable the built-in MQTT broker, and the **port** property to specify the port to be used by the broker. By default, the **enabled** property is set to `true` and the port is set to 11883. When an integration node starts, the built-in MQTT broker starts on the port that is configured by the **port** property for the MQTTServer object in the pub/sub component.

If more than one integration node is configured with the same MQTTServer port, only one MQTT broker starts. All integration nodes that are using the same MQTTServer port, use the same MQTT broker to publish their events. Subscribers that connect to the MQTT broker receive all the events that are published by the broker, unless the subscriber includes the name of the integration node in their subscriptions.

You can modify the IP address that the MQTT broker process starts by updating the value of `bind_address` in the configuration file in the *work_path*`\components\`*integrationNodeName*`\config` directory. By default, the MQTT broker process starts on the same host as the integration node; the default content of the file is `bind_address localhost`.

You can view the current configuration of the built-in MQTT broker by using the **mqsireportproperties** command. For example, use the following command to view the port that is used by the built-in MQTT broker:

```
mqsireportproperties INODE -b pubsub -o MQTTServer -n port
```

Complete the following steps to change the MQTT pub/sub broker port:

Procedure

1. Use the **mqsichangeproperties** command to stop the built-in MQTT broker.

   For example,

   ```
   mqsichangeproperties INODE -b pubsub -o MQTTServer -n enabled -v false
   ```

2. Use the **mqsichangeproperties** command to restart the built-in MQTT broker on a port other than the default port.

   For example,

   ```
   mqsichangeproperties INODE -b pubsub -o MQTTServer -n enabled,port -v true,11885
   ```

Results

When **QoS** is set to 1 or 2, the default number of messages that are held in queue is `100`. If you alter the message count, then you must alter the integration node's config file to have the following property.

```
max_queued_messages count
```

The maximum number of **QoS** 1 or 2 messages to hold in the queue above those messages that are currently in flight. The default is `100`. Set to `0` for no maximum (this value is not advised).

The value of **max_queued_messages count** is set in the configuration file. The configuration file that is used by the bipMQTT can be found under the `work_path\components\integrationNodeName \config` directory, which contains a file with the same name as Integration Node name.

For example,

If the Integration Node name is *BRK1* and you are running Windows, then a file by name `BRK1` is found under `C:\ProgramData\Application Data\IBM\MQSI\components\BRK1\config` with the following details in it:

```
bind_address localhost
```

Use the following steps to alter the built-in configuration:

1. Stop the Node.
2. Append the following line to the file.

   ```
   max_queued_messages <count>
   ```

   Where *count* is the number of inflight messages to be held.
3. Restart the Node for the built-in MQTT broker to pick up the changes.

If more complicated configurations are required, then you must use your own MQTT broker.

What to do next

By default, all event groups except the `BusinessEvents` group are enabled for MQTT transport. The `BusinessEvents` group is enabled by default for the IBM MQ pub/sub broker. The `BusinessEvents` group includes monitoring events. If you want to publish monitoring events to the built-in MQTT broker, you must configure the `BusinessEvents` group to enable MQTT publication by using the **mqsichangeproperties** command. For an example, and more information about how to specify a pub/sub broker for specific types of event, see "Configuring the publication of event messages" on page 2807.

Subscribing to event message topics

You can subscribe to topics that return messages about the configuration, state, or operational status of your integration server, integration node, or message flows.

About this task

A subscriber can send a subscription request to the pub/sub broker that specifies the event messages that the subscriber wants to receive.

Note: You can use wildcards when you subscribe to statistics reports. For example, to receive message flow statistics reports for all integration nodes and all integration servers, subscribe to the following topic:

```
topicRoot/+/StatisticsAccounting/#
```

where *topicRoot* is `IBM/IntegrationBus` for an MQTT pub/sub broker and `$SYS/Broker` for an IBM MQ pub/sub broker. For more information about how you can use wildcards, see Special characters in topics. For more information about the use of *topicRoot*, see Topic semantics and usage.

You can subscribe to IBM App Connect Enterprise messages in the following categories:

Message flow performance

If you enable message flow accounting and statistics collection for an integration node, the integration node publishes messages (JSON or XML publications) at an interval that you can control by setting the `statsInterval` property of the integration node. You can subscribe to these messages on the following topics:

- For JSON publications:

```
topicRoot/integrationNodeName/Statistics/JSON/recordType/integrationServerName/messageFlowName
```

- For XML publications:

```
topicRoot/integrationNodeName/StatisticsAccounting/recordType/integrationServerName/messageFlowName
```

For messages published by an independent integration server, subscribe to the following topics:

- For JSON publications:

```
topicRoot/integration_server/Statistics/JSON/recordType/integrationServerName/messageFlowName
```

- For XML publications:

```
topicRoot/integration_server/StatisticsAccounting/recordType/integrationServerName/messageFlowName
```

where *recordType* specifies the type of record (SnapShot or Archive), and *integrationServerName* is the value that is specified by the **--name** parameter when the IntegrationServer process is started.

Note: For an IBM MQ pub/sub broker, each message is a JMS TextMessage that contains the statistics report in XML format. If you want the message in a JMS BytesMessage format, use one of the following methods:

- Set the environment variable **MQSI_STATS_MQSTR=false**.

- Run the **mqsichangeproperties** command with the pubsub component to change the format. For example:

```
mqsichangeproperties INODE -b pubsub -o OperationalEvents/MQ -n format -v none
```

The pubsub component change overrides the use of environment variables. For more information, see Parameter values for the pubsub component.

Resource performance

If you enable resource statistics collection for one or more integration servers on an integration node, the integration node publishes messages (JSON or XML publications) at 20-second intervals. You can subscribe to these messages on the following topics:

- For JSON publications:

```
topicRoot/integrationNodeName/Statistics/JSON/Resource/integrationServerName
```

- For XML publications:

```
topicRoot/integrationNodeName/ResourceStatistics/integrationServerName
```

For messages published by an independent integration server, subscribe to the following topics:

- For JSON publications:

```
topicRoot/integration_server/Statistics/JSON/Resource/integrationServerName
```

- For XML publications:

```
topicRoot/integration_server/ResourceStatistics/integrationServerName
```

where *integrationServerName* is the value that is specified by the **--name** parameter when the IntegrationServer process is started.

Note: For an IBM MQ pub/sub broker, each message is a JMS TextMessage that contains the resource statistics report in XML format. If you want the message in a JMS BytesMessage format, use one of the following methods:

- Set the environment variable **MQSI_STATS_MQSTR=false**.
- Run the **mqsichangeproperties** command with the pubsub component to change the format.

For more information about how to interpret the resource statistics that are included in the publication, see "Viewing resource statistics data" on page 2745.

Monitoring

You can configure your message flow to emit event messages that can be used to support transaction monitoring and auditing, and business process monitoring; see "Configuring monitoring for message flows" on page 2788. To receive event messages from your message flows, subscribe to the following topics:

- For JSON publications:

```
topicRoot/integrationNodeName/MonitoringEvents/JSON/integrationServerName/applicationName/messageFlowName
```

- For XML publications:

```
topicRoot/integrationNodeName/Monitoring/integrationServerName/applicationName/messageFlowName
```

Where *applicationName* is the name of the deployment container. For example:

- Application name.
- RESTAPI name.
- Integration Service name.

Note: When you use the built-in MQTT pub/sub broker, Monitoring events are not published by default. You must enable the publication of Monitoring events to the built-in MQTT pub/sub broker.

For messages published by an independent integration server, subscribe to the following topic:

```
topicRoot/integration_server/
Monitoring/integrationServerName/applicationName/messageFlowName
```

Workload management

The message rate statistics are collected at a checkpoint that occurs every 20 seconds. The total message rate is calculated at this checkpoint, and if the total message rate exceeds the notification threshold, the out of range XML message is published. If the total message rate continues to stay above the notification threshold, then no further out of range messages are published.

If you enable the notification threshold, you can subscribe to the following topics:

- Message flow is in a library within an application:

  ```
  topicRoot/integrationNodeName/WorkloadManagement/
  AboveThreshold/integrationServerName/applicationName/libraryName/messageFlowName
  ```

- Message flow is in a library that is not within an application:

  ```
  topicRoot/integrationNodeName/WorkloadManagement/
  AboveThreshold/integrationServerName/libraryName/messageFlowName
  ```

- Message flow is directly in an application (not in a library within the application):

  ```
  topicRoot/integrationNodeName/WorkloadManagement/
  AboveThreshold/integrationServerName/applicationName/messageFlowName
  ```

- Message flow is not in an application or a library:

  ```
  topicRoot/integrationNodeName/WorkloadManagement/
  AboveThreshold/integrationServerName/messageFlowName
  ```

For messages published by an independent integration server, subscribe to the following topics:

- Message flow is in a library within an application:

  ```
  topicRoot/integration_server/WorkloadManagement/
  AboveThreshold/integrationServerName/applicationName/libraryName/messageFlowName
  ```

- Message flow is in a library that is not within an application:

  ```
  topicRoot/integration_server/WorkloadManagement/
  AboveThreshold/integrationServerName/libraryName/messageFlowName
  ```

- Message flow is directly in an application (not in a library within the application):

  ```
  topicRoot/integration_server/WorkloadManagement/
  AboveThreshold/integrationServerName/applicationName/messageFlowName
  ```

- Message flow is not in an application or a library:

  ```
  topicRoot/integration_server/WorkloadManagement/
  AboveThreshold/integrationServerName/messageFlowName
  ```

Results

Subscribers receive statistics reports only from those integration servers that are enabled to produce statistics.

IBM App Connect Enterprise event messages

Event messages are published by an integration server in response to certain conditions that occur while it is active.

The events are published on a series of system-defined topics. The body of the message contains additional information in XML format. Every message is generated in code page 1208.

Event messages are published in response to operational events, such as the publication of statistics, and business events, such as monitoring.

For more information about events, see the following topics:

- "Event message architecture" on page 2816
- "Configuring the publication of event messages" on page 2807.

Event message architecture

Integration servers publish messages on reserved topics after significant events within the integration server. By subscribing to these topics, a client is informed when these events occur.

For each topic, the type of event and message body are explained. The body of these messages is either in JSON or XML format, depending on the topic.

An event publication can contain more than one entry if the topic is the same.

The general notation of the system topics on which events are published is as follows:

```
$SYS/Broker/integrationNodeName/event_type/integrationServerName
/applications/application_name/libraries/library_name/messageflows/message_flow_name
```

For an integration server that is managed by an integration node, *integrationNodeName* is the name of the integration node that manages the integration server. For an independent integration server, specify the literal string `integration_server` in place of an integration node name.

For example:

- For an integration server that is managed by an integration node:

```
$SYS/Broker/myIntegrationNode1/Statistics/JSON/SnapShot/myIntegrationServer3/applications/
myApplicationA
/libraries/myLibraryD/messageflows/myMessageFlow6
```

- For an independent integration server:

```
$SYS/Broker/integration_server/Statistics/JSON/SnapShot/myIntegrationServer3/applications/
myApplicationA
/libraries/myLibraryD/messageflows/myMessageFlow6
```

For events that are published using MQTT as the transport, the notation is as follows:

```
IBM/IntegrationBus/integrationNodeName/event_type/integrationServerName
/applications/application_name/libraries/library_name/messageflows/message_flow_name
```

integrationNodeName
The name of the integration node that issues or raises this event, or, in the case of an independent integration server, the literal string `integration_server`.

event_type
event_type is the type of the event. The possible event types are shown in the following table:

Table 92. Types of event and how they are represented in publication topics. The table describes the event groups, event types, and the equivalent notation for *event_type*.		
Event group	**Event type**	**Notation format for *event_type***
OperationalEvents	• Resource statistics (JSON) • Resource statistics (XML) • Message flow statistics (JSON) • Message flow statistics (XML) • Workload Management	• `Statistics/JSON/Resource` • `ResourceStatistics` • `Statistics/JSON` • `StatisticsAccounting` • `WorkloadManagement`
BusinessEvents	• Monitoring	• Monitoring

Message flow statistics and resource statistics event types are published in both JSON and XML format. All other event types are published in XML format. For more information, see "Output formats for message flow accounting and statistics data" on page 2704.

When you subscribe to events, the topic structure enables you to filter the events by integration node, integration server, and type of event. For specific events, additional information is included in the topic to help filter on the specific object that raised the event. The inclusion of the string `Broker` (or for MQTT, `IntegrationBus`) at the second level of the topic hierarchy allows for future extension to additional subsystems that publish system management events through the integration node or server. For more information, see "Subscribing to event message topics" on page 2813.

Monitoring business transactions

Monitor business transactions through one or more flows in your enterprise.

Before you begin

Read the introductory information about monitoring business transactions, in "Business transaction monitoring overview" on page 2818.

About this task

You can use the App Connect Enterprise web user interface to view the results of business transactions, and to create or delete business transaction definitions.

Procedure

Complete the following steps to configure integration servers for business transaction monitoring, to create business transaction definitions, and to view the results of business transaction monitoring:

1. Configure your integration server and message flows to enable business transaction monitoring, by following the instructions in "Configuring business transaction monitoring" on page 2820.
2. Enable security for business transaction monitoring, by following the instructions in "Enabling security for business transaction monitoring" on page 2824.
3. Create a business transaction definition, by following the instructions in "Creating a business transaction definition" on page 2826.
4. Start the business transaction definition, by following the instructions in "Starting and stopping a business transaction definition" on page 2827.
5. View the status and results of business transactions, by following the instructions in "Viewing the results of business transaction monitoring" on page 2830.
6. Optional: Update an existing business transaction definition, by following the instructions in "Updating a business transaction definition" on page 2828.

7. Optional: Delete a business transaction definition, by following the steps in "Deleting a business transaction definition" on page 2830.

Business transaction monitoring overview

Business transaction monitoring involves monitoring a message across multiple message flows, so you can track and report the lifecycle of a payload message through an end-to-end enterprise transaction.

Business transaction monitoring enables you to track the outcomes of work passing through a message flow, allowing you to see which aspects of your business transaction are working correctly and which aspects are failing.

Video: Business Transaction Monitoring

This video demonstrates the capability called Business Transaction Monitoring. The video describes the key concepts and demonstrates an overview of how to configure and use Business Transaction Monitoring.

The following terms describe aspects of business transaction monitoring:

Business transaction
A set of business events or actions that form a self-contained business use case, such as the booking of an airline ticket. A business transaction might be implemented as multiple user transactions or activities.

Business transaction instance
One instance of a business transaction. Multiple instances of a business transaction can be running concurrently, such as when multiple customers are booking airline tickets.

Business transaction definition
The definition of the business transaction, which you create by using the web user interface. The business transaction definition specifies the flows that are part of the business transaction. It also specifies which monitoring events signify the start and end of the business transaction, which events show that a business transaction is in progress, and which events signify a failure in the transaction.

The business transaction definition comprises information that is specified in a business transaction policy, and configuration properties that are set in the integration server's `server.conf.yaml` file. The business transaction policy defines the events that will be monitored as part of the business transaction, and the configuration properties in the integration server's `server.conf.yaml` file specify the name of the policy and the data store.

When you create the business transaction definition in the web user interface, you specify the name of a business transaction policy and select the business events that will be monitored as part of the business transaction. If the policy already exists, it is updated automatically with the business events that you have specified. If the policy does not exist, it is created for you. The `BusinessTransactionDefinitions` section of the integration server's `server.conf.yaml` file is updated automatically with the name of the policy that you selected.

Business transaction event
A monitoring event that signifies the start, end, or failure of a business transaction or shows how the transaction is progressing. Monitoring events are defined for the message flows in your business transaction. When you create a business transaction definition, you select appropriate monitoring events and flag them as start, end, failure, or progress business transaction events. Not all monitoring events need to be flagged as business transaction events.

A *start* business transaction event typically corresponds with an input node in a message flow. This event signifies that when a message is received on that input node, the business transaction starts.

An *end* business transaction event typically corresponds with a transaction end event for an input node. This event signifies that when a message reaches the end of that flow, the business transaction is complete.

A *failure* business transaction event typically corresponds with a node that handles error processing. This event signifies that when a message reaches this node, the business transaction has failed.

A *progress* business transaction event can correspond with any type of node in a message flow. This event signifies that a message has reached a particular node in the flow.

These business transaction events are used to show the progress of a business transaction on the Monitor tab of the business transaction monitor:

- If a start event is issued for a business transaction, but the end event or failure event is not yet issued, that business transaction instance is shown as being `In progress`.
- If an end event is issued for a business transaction, that business transaction instance is shown as being `Complete`.
- If a failure event is issued for a business transaction, that business transaction instance is shown as having `Failed`.
- If an in-progress event is received after an end event, the business transaction is shown as being `Inconsistent`. Other situations can also result in an inconsistent business transaction.

You can use business transaction monitoring to track the lifecycle of a message through an end-to-end enterprise transaction. Business transaction monitoring uses events that are emitted from message flows, and a transaction is correlated across a set of events using the global transaction ID that is defined in an event. The global transaction ID can be inferred from an upstream node by using the **`Automatic`** property, or you can set it explicitly with an XPath expression.

Tip: It is not necessary to flag all monitoring events in a business transaction as business events, and doing so can adversely affect the performance of business transaction monitoring. The extent to which performance is affected depends on the number and complexity of business transaction events, and the rate at which they are being processed. When business events are emitted by the flows, they are read from MQ topics and put to an MQ queue, which is specified in the integration server's `server.conf.yaml` file (by default, this queue name is `SYSTEM.BROKER.DC.RECORD`). Performance of the business transaction monitor can be impacted if the business events are emitted to the queue faster than they can be written to the database. To improve performance, focus on flagging events that occur in error conditions, so that you can monitor when and why a business transaction might have failed.

For event monitoring, the presentation of data is event-oriented. For business transaction monitoring, presentation of data is transaction-oriented. Initially, business transaction data is presented as a summary of transactions. You can then view the details of the business transaction, the system-level transactions, and a list of events by inspecting individual transactions.

The web user interface is used to monitor business transactions. You create a *business transaction definition* to identify the message flows that form the business transaction, and define how they interact with each other. You can view the monitoring events that are defined for the message flows that make up the business transaction. Some of these events are significant to the business transaction. The business transaction definition defines how these events apply to the business transaction. Existing monitoring events can be selected as start, end, failure, and progress events in the business transaction.

A business transaction policy specifies the business events that will take part in the business transaction, and this policy forms one part of the business transaction definition. Additional configuration information, such as details of the data store that contains the business data for the business transaction, is specified in the integration server's `server.conf.yaml` configuration file. You set the database location by specifying the data store in the `server.conf.yaml` file, and the business transaction policy is updated with the new data source name as a result. It is not necessary to make any other changes to the business transaction policy; however, if you want to change any other properties in the policy, you can modify it by using an external editor. For more information, see “Creating a business transaction definition” on page 2826.

The results of business transactions can be viewed in the web user interface. You can see which instances of business transactions are in progress, which instances completed successfully, and which instances failed. Detailed information is also shown for the transaction-specific events of a business transaction.

The business transaction monitor in the web user interface has two tabs: **Configure** and **Monitor**. Use the Configure tab to create or modify a business transaction definition, and to define the business events that participate in the business transaction. Create a new business transaction definition by clicking the **Create a transaction definition** button, and specifying the business transaction policy in which the

business events are to be defined. For a monitoring event to be part of a business transaction, it must be explicitly added to the business transaction definition as a business event, by clicking the **Create an event** button in the Configure tab, and then selecting the monitoring event that you want to define as a business event. For more information, see "Creating a business transaction definition" on page 2826 and "Updating a business transaction definition" on page 2828.

Use the Monitor tab to view instances of the business transaction, and the status of those instances and associated events. To show all the latest business transactions, click the Refresh icon (). Alternatively, you can show only the instances that have a particular status, or instances that were last updated within a specified period of time. For more information, see "Viewing the results of business transaction monitoring" on page 2830.

Configuring business transaction monitoring

Configure your integration server and message flows to enable business transaction monitoring.

About this task

You configure the message flows that you want to participate in the business transaction, and create the business transaction definitions and policies that define the business events that will be monitored end-to-end throughout the business transaction. You must also configure your integration server to enable business transaction monitoring, and define the data stores that are to be used for recording and viewing the business events.

Video: Configure App Connect Enterprise to use Business Transaction Monitoring - detailed view

This video demonstrates in detail how to configure IBM App Connect Enterprise to use Business Transaction Monitoring.

Procedure

Configure your integration server and message flows to enable business transaction monitoring, by completing the following steps:

1. Configure monitoring on the message flows that are to participate in the business transaction, either by setting properties on the message flow nodes or by using a monitoring profile.

   Ensure that the correlation information is set correctly. To be part of a business transaction, a flow must have events that are correctly defined and enabled, and that contain a global transaction ID when they are emitted by the flow. The global transaction ID is a unique ID such as a reference number or an order processing number, which is used to correlate all the events for a single business transaction. The global ID can either be derived from a previous node in the flow by using the `automatic` setting, or it can be set explicitly by using an XPath expression to specify a location in the message. A global transaction ID links events from a message flow to one or more related message flows or external applications. For more information, see "Configuring monitoring event sources by using monitoring properties" on page 2788 and "Configuring monitoring event sources by using a monitoring profile" on page 236.
2. Enable monitoring for the integration servers on which the flows are running (and which will therefore be publishing the events), by setting properties in the `Monitoring/MessageFlow` section of the integration server's `server.conf.yaml` file. Set the **`publicationOn`** property to `active`, and the **`eventFormat`** property to `MonitoringEventV2`, as shown in the following example:

```
Monitoring:
  MessageFlow:
    publicationOn: 'active'           # choose 1 of : active|inactive, default is inactive
                                      # Ensure Events.BusinessEvents.MQ|MQTT is set
    eventFormat: 'MonitoringEventV2'  # choose 1 of : MonitoringEventV2|WMB
```

3. Enable business monitoring events for the integration servers on which the flows are running, by setting the **enabled** property to `true`, in the `Events/BusinessEvents/MQ` section of the integration server's `server.conf.yaml` file:

```
Events:
  BusinessEvents: # Business monitoring events
    MQ:
      policy: ''                # Specify a policy in the form {policy project}:policy if
 not using 'defaultQueueManager'
      enabled: true             # Set true or false, default is false
      format: ''                # Set string or none
      outputFormat: 'xml'       # Set to xml or json, default is xml
      publishRetryInterval: 0   # Set the retry interval (in milliseconds), to pause
 all publications and retry when publication failures are causing serious delay to the
 transaction.
```

   The integration servers that are publishing the events must point to the same queue manager that is recording the events, either by specifying the **defaultQueueManager** property or by setting the **policy** property in the `Events/BusinessEvents/MQ` section.

Install and configure a database for recording monitoring events and business transaction data, and configure the integration server to record and access the data:

4. Install and configure a DB2 or Oracle database for recording monitoring events and business transaction data, and configure a connection to the integration, as described in "Enabling ODBC connections to the databases" on page 184.

   For more information, see "Configuring databases" on page 182.

5. Run the `DataCaptureSchema.sql` script to create the monitoring events tables that are required for business transaction monitoring.

   If these tables have already been created for use by the record and replay function, you can use the same tables for business transaction monitoring events. However, if the tables do not exist, or if you want to use a different schema for business transaction monitoring from the one that you use for record and replay, you must run the `DataCaptureSchema.sql` script as described in the relevant topic:

   - "Creating and configuring a DB2 database for recording data" on page 2842
   - "Creating and configuring an Oracle database for recording data" on page 2843

   In addition to the `DataCaptureSchema.sql` script that you run as part of this configuration procedure, a file called `DataCaptureSchema_v2.sql` is stored in the same location in the file system. However, the `DataCaptureSchema_v2.sql` script can be used only if you are using V2 monitoring events exclusively, and cannot be used in any architecture that combines V1 and V2 monitoring events.

6. Run the BusinessCaptureSchema.sql script to create the database table that is required for recording and storing business transaction data, as shown in the following examples:

   - DB2:

     ```
     db2 -tvf BusinessCaptureSchema.sql
     ```

   - Oracle:

     ```
     sqlplus user/password @BusinessCaptureSchema.sql
     ```

   The script is stored in the following location, depending on your database and operating system:

   DB2

   - Windows: *install_dir*\server\ddl\db2\BusinessCaptureSchema.sql
   - Linux or UNIX: *install_dir*/server/ddl/db2/BusinessCaptureSchema.sql

   Oracle

   - Windows: *install_dir*\server\ddl\oracle\BusinessCaptureSchema.sql
   - Linux or UNIX: *install_dir*/server/ddl/oracle/BusinessCaptureSchema.sql

   where *install_dir* is the location of your IBM App Connect Enterprise installation.

   The database tables are now ready to be used for recording and storing your business transaction data.

7. Configure the Record and Replay store for the integration server, by setting properties in the Stores section of the integration server's server.conf.yaml file.

   The name of this data store must be unique. By default, business transaction definitions created in the web user interface are configured to reference a Record and Replay store called BTMDataStore, as shown in the following example:

```
Stores:
    BTMDataStore:
       dataSource: ''                    # The ODBC data source name (DSN) that is used to
 connect to the database that stores the recorded data. This property is mandatory and has
 no default value.
       schema: ''                        # The schema name that owns the database tables
 that are used for storing recorded data. This property has no default value. If no value
 is set, either the default database schema is used (if there is one), or no schema is used,
 depending on the database.
       storeMode: 'all'                  # The mode for the store to operate in. Valid
 values are record, view, and all. Default is all.
       queue: 'SYSTEM.BROKER.DC.RECORD'  # The name of the queue to which event messages
 will be published before being recorded to the database. The queue must exist.
                                          # Default is SYSTEM.BROKER.DC.RECORD. The queue
 SYSTEM.BROKER.DC.RECORD must be created manually if you use Record and Replay. The same
 queue can be specified for multiple Record and Replay stores.
                                          # Change the value of this property to distribute
 the data from multiple sources across multiple queues.
       backoutQueue: 'SYSTEM.BROKER.DC.BACKOUT'  # The name of the backout queue used by the
 recorder. Messages that cannot be processed (for example, because the specified database
 does not exist) are sent to this queue.
                                                  # Default is SYSTEM.BROKER.DC.BACKOUT.
 The queue SYSTEM.BROKER.DC.BACKOUT must be created manually if you use Record and Replay.
 If a data capture source refers to this data capture store, and no backoutQueue has been
```

```
specified, an error occurs. The same backoutQueue can be specified for multiple Record and
Replay stores.
```

a) Specify the ODBC data source name (DSN) that is used to connect to the database where the recorded data is stored, by setting the **dataSource** property.

b) Specify the schema name for the database tables that are used for storing recorded data, by setting the **schema** property.

   If you do not set this property, the default database schema is used.

c) Specify the names of the queue and backout queue to be used by the recorder, by setting the **queue** and **backoutQueue** properties. By default, these properties are set to SYSTEM.BROKER.DC.RECORD and SYSTEM.BROKER.DC.BACKOUT.

   If you specify non-default queues in the **queue** and **backoutQueue** properties, you must also specify them in the Record and Replay store in which the ODBC data source name is specified (using the **dataSource** property), and the Record and Replay store must be the one that is referenced from the business transaction definition (by default, BTMDataStore).

8. Enable Record and Replay for the integration server, by editing the server.conf.yaml file and setting the **recordReplayEnabled** property to true.

```
# Record and Replay requires a default queue manager to be associated with the integration
 server.
   RecordReplay:
     recordReplayEnabled: true           # Set to true to enable all Record and Replay
 functionality. Default is true.
```

Configure the integration server to use a default queue manager, and create the default system queues on the queue manager:

9. Configure the integration server to use a local or remote default queue manager, by specifying either the **defaultQueueManager** or **remoteDefaultQueueManager** property in the integration server's server.conf.yaml file. For example:

```
# The remoteDefaultQueueManager and defaultQueueManager properties are mutually exclusive.
 Uncomment only one of these two options.
      defaultQueueManager: 'defaultQM'            # Set non-empty string to specify a
 default queue manager
```

or

```
# The remoteDefaultQueueManager and defaultQueueManager properties are mutually exclusive.
 Uncomment only one of these two options.
      remoteDefaultQueueManager: '{myPolicyProject}:myMQEndpointPolicy'      # Specify an
 MQEndpoint policy in the format {policy project}:policy
```

10. Ensure that the SYSTEM.BROKER.DC.RECORD and SYSTEM.BROKER.DC.BACKOUT system queues have been created on the queue manager. If they do not exist, you can create them by running the iib_createqueues script, as described in “Creating the default system queues on an IBM MQ queue manager” on page 99.

Configure the required security permissions for users of the web user interface:

11. If administration security has been enabled on the integration server, ensure that users have the required permissions, by setting properties in the `Security` section of the integration server's `server.conf.yaml` file:
    - For viewing transactions, users require **`read`** access on the `DataPermissions` object and the `Permissions` object.
    - For creating and editing business transaction definitions and business transaction policies, users require **`write`** access on the `Permissions` object.
    - For starting and stopping the recording of business transactions, users require **`execute`** access on the `Permissions` object.

```
Security:
  Permissions:
    # Set Admin Security Authorization file permissions by web user role using 'read+:write
+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission
    # e.g. define the following web user roles 'viewRole' and 'adminRole'
    #viewRole:  'read+:write-:execute-'
    #adminRole: 'all+'
  DataPermissions:
    # Set Admin Security Authorization file permissions for Record and Replay web user role
 using 'read+:write+:execute+' , or 'all+'
    # '+' grants permission, '-' denies permission.  Record and Replay roles also require
 'read+' permission to be defined
    # in the Permissions section above.
    # e.g. define the following web user roles 'dataViewer', 'dataReplayer' and 'adminRole'
    #dataViewer:  'read+:write-:execute-'
    #dataReplayer: 'read+:write-:execute+'
    #adminRole: 'all+'
```

12. Ensure that the web user interface is configured. For more information, see “Configuring the IBM App Connect Enterprise web user interface” on page 87.

Enabling security for business transaction monitoring

You can restrict the users who can view and configure business transaction monitoring for an integration server, by enabling administration security and setting permissions for specified roles.

Before you begin

Read the following topics:

- “Authorizing users for administration” on page 2516
- “Monitoring business transactions” on page 2817

About this task

If you do not enable administration security, any user can complete any action against an integration node and its resources, or against an integration server and its resources. You can enable administration security and specify the authorization mode for an integration node or server, by using the **`mqsichangeauthmode`** command.

Procedure

To enable security for business transaction monitoring, complete the following steps:

1. Stop the integration node or server:
    - To stop an integration node or a managed integration server, stop the integration node by using the web user interface or by running the **`mqsistop`** command.
    - For an independent integration server, stop the integration server by stopping its **`IntegrationServer`** process. For more information, see “Stopping an integration server” on page 254.

2. Enable administration security for an integration node, or an integration server, and specify an authorization mode by using the **mqsichangeauthmode** command.

   For example, to enable administration security with the file-based authorization mode for the ACE12NODE integration node, enter the following command:

   ```
   mqsichangeauthmode ACE12NODE -s active -m file
   ```

   where `-s active` enables administration security for the integration node, and `-m file` specifies the file-based authorization mode.

   For more information, see "Enabling administration security" on page 2504.
3. Define the roles and their associated permissions:

   - If the integration node or integration server is configured to use file-based authorization (**file** mode), you define the roles and associated permissions on the integration node or integration server, by using the **mqsichangefileauth** command. For information about setting permissions for file-based authorization, see "Setting file-based permissions" on page 2530.
   - If the integration node or integration server is configured to use queue-based authorization (**mq** mode), you must create a system user ID on the operating system that is running your integration node, or integration server. You must then assign permissions to the system user ID, which is then used as a role. For information about setting permissions for queue-based authorization, see "Setting queue-based permissions" on page 2533.

   One or more web user IDs can be assigned to each role, and the permissions that were granted to the role are automatically granted to all web user IDs that are assigned to it. For more information, see "Role-based security" on page 2503 and "Managing web user accounts" on page 2510.
4. To allow users with an assigned role to view the results of business transaction monitoring on the integration server, set the required permissions for the role, using either file-based or queue-based permissions, depending on the authorization mode that is set for the integration node or integration server:

   - If you are using file-based authorization, set `read+` permission for the role for actions on the integration node or the integration server. For more information about file-based authorization, see "Setting file-based permissions" on page 2530.
   - If you are using queue-based authorization, set `+inq` permission for the role for actions on the queues SYSTEM.BROKER.AUTH and SYSTEM.BROKER.AUTH.*EG*. For more information about queue-based authorization, see "Setting queue-based permissions" on page 2533.
5. You must also set the required permissions for configuring business transaction monitoring, control the actions that users with a specified role (such as `ibmuser`) can complete on the integration node or server. Ensure that the role has the appropriate authorization to complete the required actions, as described in "Controlling access to data and resources in the web user interface" on page 2544.

   To change permissions for an integration node or server that is using file-based authorization, see "Setting file-based permissions" on page 2530. To change permissions for an integration node or server that is using queue-based permissions, see "Setting queue-based permissions" on page 2533.
6. Create a web user account by using the **mqsiwebuseradmin** command, and specify a role for the account. This account is the one that you will use to log on to the web user interface for using business transaction monitoring.

   For more information, see "Managing web user accounts" on page 2510.
7. Start the integration node or server:

   - Start an integration node or managed integration server by using the web user interface or by running the **mqsistart** command.
   - For an independent integration server, start the integration server by using the **IntegrationServer** command. For more information, see "Starting an integration server" on page 250.

What to do next

For information about how to configure business transaction monitoring, see “Configuring business transaction monitoring” on page 2820. For information about how to view the results of business transaction monitoring, see “Viewing the results of business transaction monitoring” on page 2830.

Creating a business transaction definition

Create a business transaction definition in the App Connect Enterprise web user interface, by specifying message flows and defining business events in an associated business transaction policy.

Before you begin

- Read the information in “Monitoring business transactions” on page 2817 and “Business transaction monitoring overview” on page 2818.
- Configure business transaction monitoring, as described in “Configuring business transaction monitoring” on page 2820.
- Ensure that IBM MQ is installed.

About this task

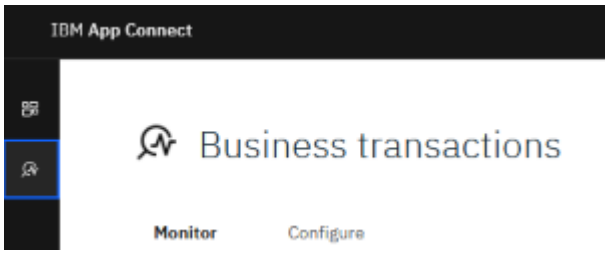
The business transaction definition contains information about the flows that participate in the transaction and the events that are monitored during the transaction. This definition includes information such as which events mark the beginning and end of a transaction, which events indicate that the transaction is in progress, and which events correspond to a failure in the transaction.

Procedure

To create a business transaction definition, complete the following steps:

1. Access the web user interface, as described in “Accessing the web user interface” on page 89.
2. Click the **Business transactions** icon to display the Business Transactions view for the integration server or integration node.

   The **Business transactions** icon is located in the side navigation links in the web user interface.

3. In the Business Transactions view, click the **Configure** tab.
4. In the Configure tab, click **Create a transaction definition**.

   The **Create a business transaction definition** panel is displayed.
5. Enter a name for the business transaction in the **Name** field.

   The name of the business transaction must be unique for the integration server. The name can contain only letters, numbers, and underscore (_) characters.
6. In the **Policy project** field, either select the policy project that will contain the business transaction policy for the new business transaction definition, or click **Create new policy project** and enter the name of a a new policy project.
7. In the **Policy** field, select the name of the business transaction policy in which the business events will be defined, or click **Create new policy** and enter the name of a new policy.

8. Click **Create** to create the new business transaction definition and, if applicable, a new business transaction policy.

   A new business transaction definition is created in the `stopped` state.
9. Add a new business event to the business transaction policy, by clicking **Create an event**.

   The **Create a business event** panel is displayed.
10. Enter the name of the business event in the **Name** field, and then select the event type that identifies the start, end, failure, or progress of the event, from the **Type** field:
    - A *start* business transaction event signifies that when a message is received on that node, the business transaction starts. You can choose any monitoring event to represent the start of a business transaction.
    - An *end* business transaction event signifies that when a message is received on that node, the business transaction is complete. You can choose any monitoring event to represent the end of a business transaction.
    - A *failure* business transaction event typically corresponds with a node that handles error processing. This event signifies that when a message reaches this node, the business transaction has failed.
    - A *progress* business transaction event can correspond with any type of node in a message flow. This event is used to monitor the progress of a transaction instance and signifies that a message has reached a particular node in the flow.

    Each business transaction definition requires a start event and an end event (in addition to any progress or failure events), and a warning message is displayed if they have not been defined. A warning message is also displayed if the flows have no monitoring events defined, or if no global transaction ID has been specified for the events.
11. When you have specified the name and type of the business event, specify the monitoring event that will be used for the business event, by selecting an integration, library, message flow, and monitoring event source, in the **Create a business event** panel. Then click **Create**.

    The business event is defined in the business transaction policy, and is displayed in the event definitions table for the business transaction definition.
12. Repeat steps “9” on page 2827, “10” on page 2827, and “11” on page 2827, until you have defined all the business events for your business transaction definition.

What to do next

Start the business transaction definition, as described in “Starting and stopping a business transaction definition” on page 2827. You can then view the results of a business transaction on the Monitor tab, as described in “Viewing the results of business transaction monitoring” on page 2830.

Starting and stopping a business transaction definition

Start or stop a business transaction definition by using the App Connect Enterprise web user interface.

Before you begin

Read the following topics:

- “Monitoring business transactions” on page 2817
- “Configuring business transaction monitoring” on page 2820

About this task

Before you can view the results of business transaction monitoring, you must start the business transaction definition. Before you can delete a business transaction definition, or make changes to its events, it must be stopped.

Procedure

- Start a business transaction definition by completing the following steps:
  1. Access the web user interface, as described in "Accessing the web user interface" on page 89.
  2. Click the **Business transactions** icon () to display the Business Transactions view for the integration server or integration node.
  3. In the Business Transactions view, click the **Configure** tab.
  4. Select the business transaction definition that you want to start, from the business transactions list on the left side of the panel.
  5. Click the actions icon (⋮) to display a list of options for the selected business transaction definition, and then click **Start monitoring**.
- Stop a business transaction definition by completing the following steps:
  1. Access the web user interface, as described in "Accessing the web user interface" on page 89.
  2. Click the **Business transactions** icon () to display the Business Transactions view for the integration server.
  3. In the Business Transactions view, click the **Configure** tab.
  4. Select the business transaction definition that you want to stop, from the business transactions list on the left side of the panel.
  5. Click the actions icon (⋮) to display a list of options for the selected business transaction definition, and then click **Stop monitoring**.

Updating a business transaction definition

Update the business events in a business transaction definition by using the web user interface or a text editor to modify the associated business transaction policy.

Before you begin

Before you update a business transaction definition, ensure that it is stopped, by following the instructions in "Starting and stopping a business transaction definition" on page 2827.

About this task

You can update the business events in a business transaction definition by using the web user interface. Alternatively, you can use a text editor to modify the business transaction policy in which the events for the business transaction are defined. The business transaction policy is typically created in the web user interface when the business transaction definition is created, as described in "Creating a business transaction definition" on page 2826.

If you move the policy from one deployment location to another, ensure that the integration node and server details for the business events are still correct for the new deployment location. For more information about the format of business transaction policies, see "Example business transaction policy" on page 2833.

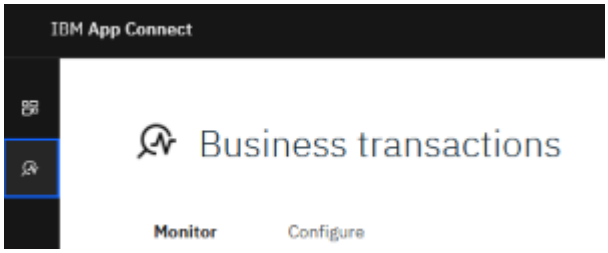
Procedure

Use the web user interface to update the business events in a business transaction definition, by completing the following steps:

1. Access the web user interface, as described in "Accessing the web user interface" on page 89.

2. Click the **Business transactions** icon to display the Business Transactions view for the integration server or integration node.

   The **Business transactions** icon is located in the side navigation links in the web user interface.



3. In the Business Transactions view, click the **Configure** tab.
4. Select the business transaction definition that you want to update, from the list on the left side of the panel.

   The business events that have been defined for the selected business transaction definition (if any) are displayed.

You can add, update, or delete business events in the selected business transaction definition, by completing the following steps:

5. Optional: **Add a business event** to the business transaction definition by completing the following steps:

   a) Click **Create an event**.

      The **Create a business event** panel is displayed.

   b) Enter the name of the business event in the **Name** field, and then select the event type that identifies the start, end, failure, or progress of the event, from the **Type** field:

      - A *start* business transaction event signifies that when a message is received on that node, the business transaction starts. You can choose any monitoring event to represent the start of a business transaction.
      - An *end* business transaction event signifies that when a message is received on that node, the business transaction is complete. You can choose any monitoring event to represent the end of a business transaction.
      - A *failure* business transaction event typically corresponds with a node that handles error processing. This event signifies that when a message reaches this node, the business transaction has failed.
      - A *progress* business transaction event can correspond with any type of node in a message flow. This event is used to monitor the progress of a transaction instance and signifies that a message has reached a particular node in the flow.

   c) When you have specified the name and type of the business event, specify the monitoring event that will be used for the business event, by selecting an integration, library, message flow, and monitoring event source, in the **Create a business event** panel. Then click **Create**.

      The business event is added to the business transaction policy, and is displayed in the event definitions table for the business transaction definition.

6. Optional: **Update a business event** by completing the following steps:

   a) Click the Edit icon ( ) in the row that contains the details of the event that you want to update.

      The **Edit a business event** panel is displayed.

   b) Modify the details that you want to update, such as the name or type of the business event, or the details of the monitoring event that will be defined as a business event, as described in step “5” on page 2829. When you have finished making the required changes, click **Save**.

7. Optional: **Delete a business event** from the business transaction definition by clicking the delete icon ( ) in the row that contains the details of the event that you want to delete, and then click the **Delete** button to confirm.

What to do next

Start the business transaction definition, as described in "Starting and stopping a business transaction definition" on page 2827. You can then view the results of a business transaction on the Monitor tab, as described in "Viewing the results of business transaction monitoring" on page 2830.

Deleting a business transaction definition

Delete a business transaction definition and its associated policy by using the App Connect Enterprise web user interface.

Before you begin

Before you delete a business transaction definition, ensure that it is stopped, by following the instructions in "Starting and stopping a business transaction definition" on page 2827.

About this task

When you delete a business transaction definition, you can choose whether to delete the associated business transaction policy, if there is one. However, if the policy is in use by other business transaction definitions, only the selected business transaction definition is deleted and the policy is retained.

Procedure

To delete a business transaction definition, complete the following steps:

1. Access the web user interface, as described in "Accessing the web user interface" on page 89.
2. Click the **Business transactions** icon ( ) to display the Business Transactions view for the integration server or integration node.
3. In the Business Transactions view, click the **Configure** tab.
4. Select the business transaction definition that you want to delete, from the list on the left side of the panel.
5. Click the actions icon ( ) to display a list of options for the selected business transaction definition, and then click **Delete**.
6. Confirm the deletion of the business transaction definition, by clicking **Delete** in the confirmation message.

   If the business transaction definition has a business transaction policy attached to it, you are prompted to delete that policy as long as it is not being used by other business transaction definitions. If an attached business transaction policy is in use by other business transaction definitions, it is retained when the selected business transaction definition is deleted.

Results

The business transaction definition and policy are removed from the Business Transactions view, but the business transaction results remain in the database until you remove them by using another method.

Viewing the results of business transaction monitoring

View the results of business transaction monitoring in the App Connect Enterprise web user interface.

Before you begin

- Read the information in "Monitoring business transactions" on page 2817 and "Business transaction monitoring overview" on page 2818.
- Create a business transaction definition, as described in "Creating a business transaction definition" on page 2826.

About this task

To see the results of the business transaction, complete the following steps:

Procedure

1. In the App Connect Enterprise web user interface, click the **Business Transactions** icon ().
   The Business Transactions window is displayed, containing a **Monitor** tab and a **Configure** tab.
2. Click the **Monitor** tab.
3. Select the required business transaction definition from the displayed list.

   The data for the selected business transaction is presented in charts and in a results table. The donut chart shows the total number of instances of the business transaction, and the number of instances that are in each state (In progress, Complete, Inconsistent, and Failed). You can display the data for transaction instances that have occurred during a specified time period, from the last 15 minutes to the last 365 days, by selecting a value from the pull-down menu above the donut chart.

   You can also choose whether to view a chart showing the number of transactions in each state, or a chart showing the duration of transactions in each state, by selecting either **Number of transactions** or **Transaction duration** from the pull-down menu. The **Number of Transactions** chart shows the number of instances of the business transaction in each state, by date. The **Transaction duration** chart shows the duration of the business transaction instances in each state, by date. In both charts, you can display the transaction instances that occurred within a specific time frame, from the last 15 minutes to the last 365 days, by selecting a value from the pull-down menu. A maximum of 1000 transaction instances can be displayed.

   The results table shows the transaction ID of each individual transaction instance, its status, the time that it started, when it was last updated, and the duration. You can reorder the information in the table by clicking the heading above a column. For example, to arrange the transaction instances in the table based on their status, click the **Transaction status** column heading.

   By default, all instances of the selected business transaction in the last 365 days are displayed, including those that have failed, those that have completed successfully, those that are still in progress, and those that have been marked as inconsistent. If a transaction instance has a status of `Inconsistent`, it means that a problem occurred during the transaction that prevented it from being resolved as `In progress`, `Complete`, or `Failed`. For example, if a transaction passed through your flows twice, two start events and two end events would be received, and the transaction would be marked `Inconsistent`.

4. Optional: You can filter the data in the results table to include a subset of transaction instances, based on their status, transaction ID, start time, and the date of their last update, by clicking the **Filter** button above the table.

   The **Build a query** dialog is displayed, in which you can add one or more of the following conditions to the filter:

   - **Transaction status:** To display only the transaction instances that are in a particular state, complete the following steps:
     a. In the **Build a query** dialog, click **Add condition**
     b. Select **Transaction status** from the first pull-down menu, then select the required status from the subsequent menus.
     c. Click **Apply**.

     The results table is updated to display only the transaction instances with the specified transaction status, which can be one or more of `Complete`, `In progress`, `Inconsistent`, or `Failed`.
   - **Transaction ID:** To display only the transaction instance with a specified ID, complete the following steps:
     a. In the **Build a query** dialog, click **Add condition**
     b. Select **Transaction ID** from the first pull-down menu, then define the search criteria for the required transaction IDs by specifying details in the subsequent menus.
     c. Click **Apply**.

     The results table is updated to display only the transaction instances that match the specified condition. For example, if you specify a transaction ID of `trans-id1`, all transaction IDs that include that string are displayed, including `trans-id1`, `trans-id10`, `trans-id100`, and `trans-id1000`, if they exist.
   - **Start time:** To display only transaction instances that started before or after a specified time, complete the following steps:
     a. In the **Build a query** dialog, click **Add condition**
     b. Select **Start time** from the first pull-down menu, then select either `Before` or `After`, then specify the required date and time from the subsequent menus.
     c. Click **Apply**.

     The results table is updated to display only the transaction instances that started during the specified time period.
   - **Last updated:** To display only the instances that were last updated before or after a specified time, complete the following steps:
     a. In the **Build a query** dialog, click **Add condition**
     b. Select **Last updated** from the first pull-down menu, then select either `Before` or `After`, and then specify the required date and time from the subsequent menus.
     c. Click **Apply**.

     The results table is updated to display only the transaction instances that were last updated during the specified time period.

5. To see the latest updates at any time, click the Refresh icon ().
6. You can display more detailed information about any individual transaction instance, by clicking the transaction ID in the table.

   A table is displayed, containing information about each of the business events that were received during the transaction instance. This information includes the time that the business event was received, the event name, the event type, the message flow, and the integration.

   You can display more detailed information about the monitoring event, including any message payload that was included in the event, by clicking on the timestamp.

Example business transaction policy

Example business transaction policy file.

You can update the business events in a business transaction definition by using the web user interface or a text editor to modify the associated business transaction policy.

If you move the policy from one deployment location to another, ensure that the integration node and server details for the events are still correct for the new deployment location.

The following example shows the type of information that might be contained in a business transaction policy:

```
{
  "eventSources": {
    "startEventSources": [
      {
        "name": "Start overall business transaction - FILE",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "fileinput.transaction.Start"
      },
      {
        "name": "Start overall business transaction - JMS",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "jmsinput.transaction.End"
      }
    ],
    "progressEventSources": [
      {
        "name": "JMS processing step complete",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "jmsinput.transaction.End"
      },
      {
        "name": "File processing step complete",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "fileinput.transaction.End"
      },
      {
        "name": "Begin notification",
        "integrationNodeName": null,
        "integrationServerName": "IndependentServerNotify",
        "integrationName": "NotifyApp",
        "libraryName": "CommonSubflows",
        "flowName": "NotifyFlow",
        "eventSourceAddress": "NotifyMQInput.terminal.out"
      }
    ],
    "endEventSources": [
      {
        "name": "Complete - notification sent",
        "integrationNodeName": null,
        "integrationServerName": "IndependentServerNotify",
        "integrationName": "NotifyApp",
        "libraryName": "CommonSubflows",
        "flowName": "NotifyFlow",
        "eventSourceAddress": "NotifyMQInput.transaction.End"
      }
    ],
    "failureEventSources": [
      {
```

```
        "name": "Fail - rollback business transaction - JMS",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "jmsinput.transaction.Rollback"
      },
      {
        "name": "Fail - rollback business transaction - FILE",
        "integrationNodeName": "DemoNode",
        "integrationServerName": "default",
        "integrationName": "IN-OUT",
        "libraryName": null,
        "flowName": "INOUT",
        "eventSourceAddress": "fileinput.transaction.Rollback"
      }
    ]
  }
}
```

For more information, see “Monitoring business transactions” on page 2817 and “Creating a business transaction definition” on page 2826.

Recording, viewing, and replaying data

Record data that is processed by a message flow and replay the data.

About this task

If you configured your message flow to emit event messages, the monitoring events publish selected message data, which you can then view or replay (resubmit for processing). The record function subscribes to the published monitoring data, and stores the data in a database. You can then view the data, or replay it by resending the message to an IBM MQ queue through the web user interface or by using the administration REST API.

For more information about recording and replaying data, see “Record and replay” on page 2835. For more information about monitoring events, see “Configuring monitoring for message flows” on page 2788.

To record, view, and replay data, follow these steps:

Procedure

1. Configure IBM App Connect Enterprise to record data. For more information, see “Recording data” on page 2837.
2. Configure the monitoring event points in the message flow.

   For more information, see “Configuring monitoring for recording” on page 2847.
3. Deploy your message flow.

   For more information, see “Deployment rules and guidelines” on page 2465.
4. Configure IBM App Connect Enterprise to view recorded data through the web user interface or the administration REST API. For more information, see “Viewing recorded data” on page 2849.
5. Replay data to an IBM MQ queue by using the web user interface or the administration REST API. For more information, see “Replaying data” on page 2850.
6. (Required if security is enabled.) If you want to view or replay recorded data through the web user interface, you must create a web user account by using the **mqsiwebuseradmin** command. See “Managing web user accounts” on page 2510 and command.
7. If you want to control users' ability to record, view, and replay data, you must enable integration node administration security, and also configure security for recording. For information, see “Enabling administration security” on page 2504 and “Enabling security for record and replay” on page 2835.
8. If you want to use different integration nodes or different independent integration servers for deploying your message flows and for recording data from those message flows to a database, you

must configure a publish/subscribe relationship between the integration nodes or integration servers. For more information, see “Using multiple integration nodes and multiple independent integration servers for record and replay” on page 2851.

Record and replay

For audit or problem determination purposes, you can record data to a database, then view it, and replay it.

Video: Record and Replay

This video describes the key concepts of Record and Replay, and demonstrates an overview of how to configure and use Record and Replay.

If you need an audit record of messages that pass through the message flows, you can record those messages. You might also want to record messages if you need to keep a history of messages for development and test purposes, or to help in problem determination. For more information, see “Recording data” on page 2837.

To determine which data is recorded, you must configure monitoring on the message flow. For more information, see “Configuring monitoring for recording” on page 2847. Data is stored in a database, which can be an Oracle, Microsoft SQL Server, or DB2 database. For more information, see “Creating and configuring a database for recording data” on page 2841. You must create a data source, then use a policy to define the data source name to use when recording data.

After you have recorded data, you can view it in several ways. By using the web user interface, you can view a list of recorded messages, or you can view details of a specific message. You can also view recorded data by using the IBM App Connect Enterprise REST API. For more information, see “Viewing recorded data” on page 2849.

You can replay a message to an IBM MQ queue by using the web user interface or the REST API. You can also replay the message to another message flow for testing or problem determination. For more information, see “Replaying data” on page 2850.

Enabling security for record and replay

You can restrict the users who can view and replay data for an integration server manged by an integration node, or an independent integration server, by enabling administration security and setting permissions for specified roles.

Before you begin

Read the following topics:

- “Authorizing users for administration” on page 2516
- “Recording data” on page 2837

About this task

If you do not enable administration security, any user can complete any action against an integration node and its resources, or an integration server and its resources. You can enable administration security and specify the authorization mode for an integration node, or an integration server by using the **mqsichangeauthmode** command.

Procedure

To enable security for record and replay, complete the following steps:

1. Stop the integration node or integration server:
   - To stop an integration node or an integration server managed by an integration node, stop the integration node by using the web user interface or by running the **mqsistop** command.
   - For an independent integration server, stop the integration server by stopping its **IntegrationServer** process. For more information, see "Stopping an integration server" on page 254.
2. Enable administration security for an integration node, or an integration server, and specify an authorization mode by using the**mqsichangeauthmode** command.

   For example, to enable administration security with the file-based authorization mode for the ACE11NODE integration node, enter the following command:

   ```
   mqsichangeauthmode ACE11NODE -s active -m file
   ```

   where `-s active` enables administration security for the integration node, and `-m file` specifies the file-based authorization mode.

   For more information, see "Enabling administration security" on page 2504.
3. Define the roles and their associated permissions:
   - If the integration node or integration server is configured to use file-based authorization (**file** mode), you define the roles and associated permissions on the integration node or integration server, by using the **mqsichangefileauth** command. For information about setting permissions for file-based authorization, see "Setting file-based permissions" on page 2530.
   - If the integration node or integration server is configured to use queue-based authorization (**mq** mode), you must create a system user ID on the operating system that is running your integration node, or integration server. You must then assign permissions to the system user ID, which is then used as a role. For information about setting permissions for queue-based authorization, see "Setting queue-based permissions" on page 2533.

   One or more web user IDs can be assigned to each role, and the permissions that were granted to the role are automatically granted to all web user IDs that are assigned to it. For more information, see "Role-based security" on page 2503 and "Managing web user accounts" on page 2510.
4. To allow users with an assigned role to run record and replay queries on the integration server, set the required permissions for the role, using either file-based or queue-based permissions, depending on the authorization mode that is set for the integration node or integration server:
   - If you are using file-based authorization, set `read+` permission for the role for actions on the integration node or the integration server. For more information about file-based authorization, see "Setting file-based permissions" on page 2530.
   - If you are using queue-based authorization, set `+inq` permission for the role for actions on the queues SYSTEM.BROKER.AUTH and SYSTEM.BROKER.AUTH.*EG*. For more information about queue-based authorization, see "Setting queue-based permissions" on page 2533.
5. You must also set the required permissions for recording to control the record and replay actions that users with a specified role (such as `ibmuser`) can complete on the integration node or integration server. Ensure that the role has the appropriate authorization to complete the required actions, as described in "Controlling access to data and resources in the web user interface" on page 2544.

   To change permissions for an integration node, or integration server that is using file-based authorization, see "Setting file-based permissions" on page 2530. To change permissions for an integration node, or integration server that is using queue-based permissions, see "Setting queue-based permissions" on page 2533.
6. Create a web user account by using the **mqsiwebuseradmin** command, and specify a role for the account. This account is the one that you will use to log on to the web user interface for viewing and replaying data.

   For more information, see "Managing web user accounts" on page 2510.

7. Start the integration node or integration server:
   - Start an integration node or an integration server managed by an integration node, by using the web user interface or by running the **mqsistart** command.
   - For an independent integration server, start the integration server by using the **IntegrationServer** command. For more information, see “Starting an integration server” on page 250.

What to do next

To view data that has been recorded, see “Viewing recorded data” on page 2849. To replay data that has been recorded, see “Replaying data” on page 2850.

Recording data

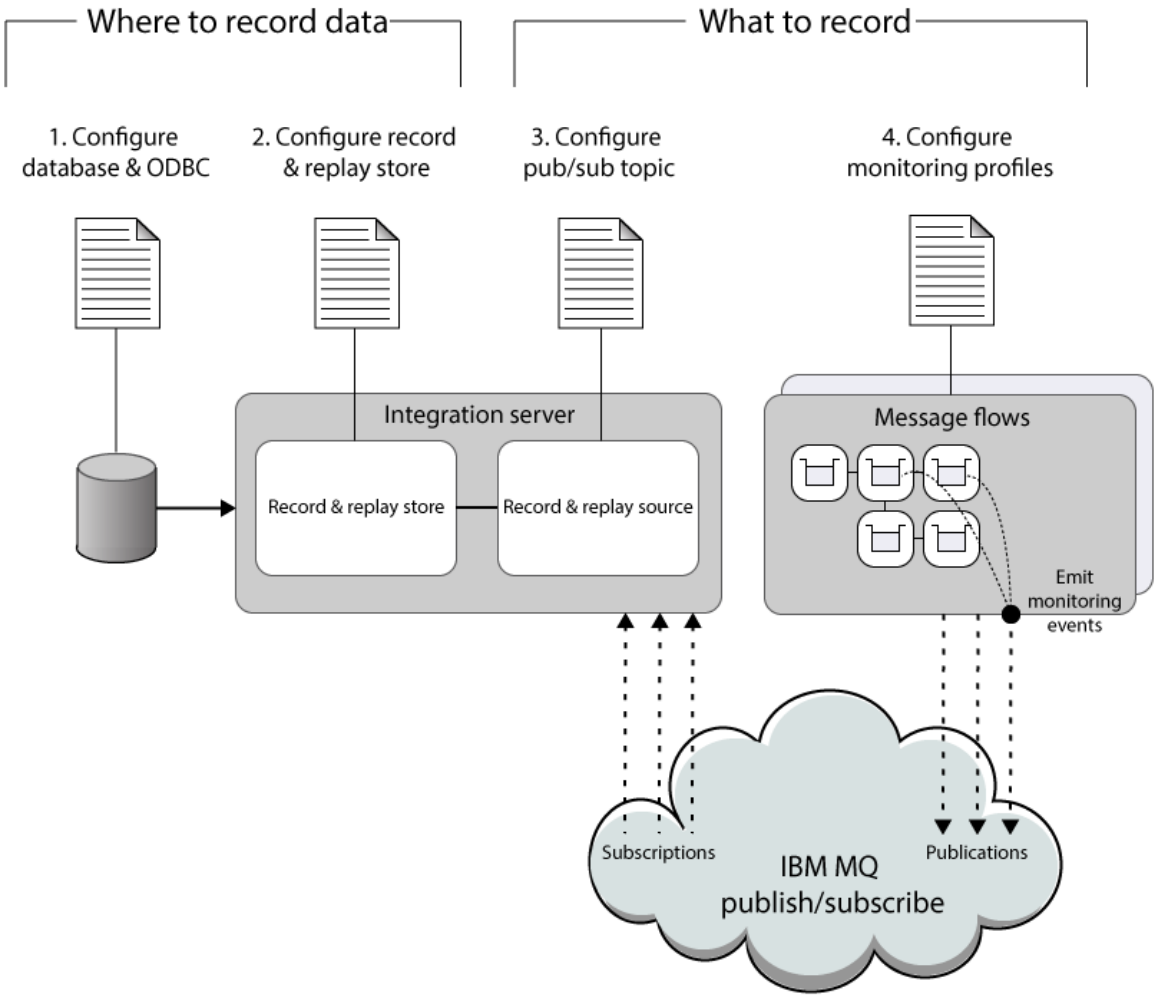
Record data that is flowing through a message flow.

Before you begin

Ensure that the message flow for which you want to record data has been deployed. For more information, see “Deployment rules and guidelines” on page 2465.

About this task

You can record data to a database for audit purposes, or to help with problem determination. To record data, you must identify the source of the data that you want to record and the place that you want to record it to. The steps that you take to record data are shown in the following diagram:



Procedure

Complete the following tasks to set up data recording. The sequence of these tasks is important. If they are not completed exactly as shown, BIP message BIP2194 is generated on startup.

1. Create and configure your database, and define an ODBC definition for the data source name (DSN). Specify an ID and password for your integration node to use when connecting to the database. See “Creating and configuring a database for recording data” on page 2841.
2. Configure the record and replay store for your integration server. See “Configuring data recording” on page 2839.

   Your record and replay topology can include more than one independent integration server or integration node. See “Using multiple integration nodes and multiple independent integration servers for record and replay” on page 2851.

3. Specify a publish/subscribe topic that identifies the source of the data that you want to record, by modifying the **topic** property of your record and replay source in the `server.conf.yaml` file for the integration server. For more information, see "Configuring data recording" on page 2839.

   Test that your subscription to the topic was successful by retrieving the subscriptions on the queue manager for your integration server.

   Use the **runmqsc** command to check the subscription. Complete the following steps:

   a. At a command prompt, type `runmqsc` *qmName*, where *qmName* is your queue manager name.

   b. To display all the queue manager subscriptions, type `dis sub(*)`

   c. Check that the topic name is returned in the list of subscription topics, for example SUB(*integrationServerName*:myTopic)

   d. To exit the **runmqsc** environment, type end

4. To generate the data that you want to record, configure monitoring on your message flows. See "Configuring monitoring for recording" on page 2847.

What to do next

- You can enable security for recording data by following the instructions in "Enabling security for record and replay" on page 2835.
- After you have recorded data, you can view it. For more information, see "Viewing recorded data" on page 2849.
- You might want to turn off recording for a particular integration server temporarily (for performance reasons, for example). For more information, see "Disabling and enabling Recording" on page 2853.
- For more information about tuning recording, See "Tuning data recording" on page 2848.
- If you use different integration nodes or independent integration servers for deploying your message flows and for recording data from those message flows to a database, then you must configure a publish/subscribe relationship between the integration nodes or integration servers . For more information, see "Using multiple integration nodes and multiple independent integration servers for record and replay" on page 2851.

Configuring data recording

Recording of data is enabled by default but takes effect only if you configure it by modifying the `server.conf.yaml` file for the integration server.

Before you begin

You must associate a queue manager with the independent integration server or the integration node that is managing integration servers. For more information, see "Creating an integration node" on page 114 and command.

About this task

You can modify the `server.conf.yaml` file to configure recording for your integration server.

Procedure

Modify the `server.conf.yaml` file to configure recording for your integration server by completing the following steps:

1. Stop the integration server. See "Stopping an integration server" on page 254.
2. Open the `server.conf.yaml` configuration file for your integration server by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are invalid in YAML and

would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
The properties that you need to set are in the **RecordReplay** section of the `server.conf.yaml` configuration file:

```
RecordReplay:
  recordReplayEnabled: true  # Set to true to enable all Record and Replay functionality.
 Default is true.
```

3. Set the **recordReplayEnabled** property to `true`.
4. Copy and customize the StoreTemplate section for each record and replay store that you want to create. For example:

```
RecordReplay:
....
  Stores:
    ..
    MBRECORD_store:
      dataSource: 'MBRECORD' # The ODBC data source name (DSN) that is used to connect to the
 database that stores the recorded data. This property is mandatory and has no default value.
      schema: 'myschema'                       # The schema name that owns the database
 tables that are used for storing recorded data. This property has no default value. If no
 value is set, either the default database schema is used (if there is one), or no schema is
 used, depending on the database.
      storeMode: 'all'  # The mode for the store to operate in. Valid values are record,
 view, and all. Default is all.
```

   a) Change the heading **StoreTemplate** to the name of your store. For example, `MBRECORD_store`.
   b) Set the **dataSource** property to the ODBC name of your database. For example, `MBRECORD`.
   c) Set the **schema** property to the name of your schema for the database tables. For example, `myschema`.
   d) Set the **storeMode** property to `all`.

   Specifying `all` for the **storeMode** property lets you use the store for recording and viewing/replaying.

   If you only want to use the store for recording:

   a. Set the **storeMode** property to `record`.
   b. Create another store, perhaps on a different integration server, for viewing/replaying.
   c. Set the **storeMode** property of the new store to `view`.

   There are other store properties you can set. For more information, see “Tuning data recording” on page 2848.
5. Copy and customize the SourceTemplate section for each record and replay source that you want to create. For example:

```
RecordReplay:
....
  Sources:
    ..
    MBRECORD_source:
      topic: '$SYS/Broker/[nodename]/Monitoring/[servername]/[applicationname]/[flowname]'  #
 Sets the subscription topic that is used for business-level monitoring of a message flow.
```

```
      store: 'MBRECORD_store' # The Record and Replay store that is used to configure record
and replay for the message flows specified in the topic property. Multiple instances of
Record and Replay source can refer to one instance of a Record and Replay store.
```

a) Change the heading **SourceTemplate** to the name of your source. For example, `MBRECORD_source`.

b) In the **topic** property, set the *nodename* to the name of your integration node.

c) In the **topic** property, set the *servername* to the name of your integration server.

d) In the **topic** property, set the *applicationname* to the name of your application.

e) In the **topic** property, set the *flowname* to the name of your message flow.

f) Set the **store** property to the name of your record and replay store. For example, `MBRECORD_store`.

A durable subscription is created for each source and is created with an id of *nodename*:*servername*:*sourcename*. If multiple independent integration servers share the same queue manager, you must ensure that each subscription has unique values for *nodename*, *servername* and *sourcename*. If you delete a source, you must manually delete the durable subscription for that source to avoid messages being published to the record and replay store's queue. For more information, see Deleting a local subscription in the IBM MQ product documentation.

6. Start the integration server. See "Starting an integration server" on page 250.

What to do next

You are now ready to start recording, viewing, and replaying your data. See "Recording data" on page 2837, "Viewing recorded data" on page 2849, and "Replaying data" on page 2850.

Creating and configuring a database for recording data

To record data, create a database and configure an ODBC definition to it. Configure your integration server so that it can connect to the database.

Before you begin

Read the concept topic "Record and replay" on page 2835. For an overview of tasks that you must complete to record data, see "Recording data" on page 2837.

About this task

You can record data to a database for audit purposes, or to help with problem determination.

A script is provided with IBM App Connect Enterprise that you can use to create the database and database tables. You can run this script unmodified, or you can customize it. The script creates a database called MBRECORD with a default schema. After creating the database, you create an ODBC definition to identify the data source name (DSN) for the database. You use the **mqsisetdbparms** command to set a user identifier and password for the integration server to use when connecting to the database. The script creates some tables that are not currently used, and are reserved for future use, such as WMB_EVENT_FIELDS and WMB_EVENT_TYPES.

You can record data to DB2, Microsoft SQL Server, and Oracle databases. For information about how to create and configure your database, see the following topics:

- "Creating and configuring a DB2 database for recording data" on page 2842
- "Creating and configuring an Oracle database for recording data" on page 2843
- "Creating and configuring a Microsoft SQL Server database for recording data" on page 2844
- "Using Integrated Windows Authentication when recording data with a Microsoft SQL Server database" on page 2845

What to do next

Continue to follow the steps for recording data; see "Recording data" on page 2837.

Creating and configuring a DB2 database for recording data

To record data to a DB2 database, create the database and configure an ODBC definition to it. Configure your integration server so that it can connect to the database.

Before you begin

Read the following topics:

- "Record and replay" on page 2835
- "Recording, viewing, and replaying data" on page 2834
- "Creating and configuring a database for recording data" on page 2841

Procedure

1. Use the script that is provided with IBM App Connect Enterprise to create and configure a DB2 database to store your recorded data. Note that the script creates some tables that are not currently used, and are reserved for future use, such as WMB_EVENT_FIELDS and WMB_EVENT_TYPES.
   a) Locate the script for your operating system:
      - Windows: *install_dir*`\server\ddl\db2\DataCaptureSchema.sql`
      - Linux or UNIX: *install_dir*`/server/ddl/db2/DataCaptureSchema.sql`

      install_dir is the location of your IBM App Connect Enterprise installation.
   b) Optional: To specify your own database or schema, customize the provided `DataCaptureSchema` script, and save your changes.

      If you modify the SQL to specify a particular schema, you must also set the same schema name in the `server.conf.yaml` file. See "Configuring data recording" on page 2839.

      You might also want to edit the script for the following reasons:

      - If you ran the script, and want to run it again, you must drop the database MBRECORD first. Insert the command `drop database MBRECORD` before the line that reads `create database MBRECORD`.
      - The maximum message body size that you can record (after encoding has taken place) is 5 MB. The default size is 5 MB, but you can increase this size by editing the script to make the value in the WMB_BINARY_DATA.DATA column larger.
   c) At a command line, navigate to the script location and run it.

      On Windows, use a **DB2 Command Window** to ensure that the command environment is set up correctly. Click **Start** > **IBM DB2** > ***databaseInstance*** > **Command Line Tools**, and select **Command Window**, where *databaseInstance* is the DB2 installation name.

      On Linux and UNIX, a script called `db2profile` is provided for setting up the environment; for more information, see "Running database setup scripts before starting an integration node" on page 83.

      When the command environment is set up, you can run the script.

      For example, on Windows, UNIX, or Linux, enter the following command:

      ```
      db2 -tvf DataCaptureSchema.sql
      ```
2. Create an ODBC definition for the database.

   If you used the supplied script to create your database without modifications, create an ODBC definition for the database called MBRECORD, with MBRECORD as the data source name (DSN). For more information, see "Enabling ODBC connections to the databases" on page 184.

3. Use the **mqsisetdbparms** command to set a user identifier and password for the integration server to use when connecting to the database; for example:

```
mqsisetdbparms -w workDir -n dataSourceName -u userID -p password
```

   - *workDir* is the name of the work directory for your integration server.
   - *dataSourceName* identifies the database to which you want to record data.
   - *userID* and *password* specify the user identifier and the password that the integration server uses to connect to the database.
4. To ensure that the changes to the **mqsisetdbparms** command take effect, restart the integration server.

   For more information, see "Starting an integration server" on page 250.
5. Test the connection to your database by using the **mqsicvp** command.

   For more information, see command.

What to do next

Continue to follow the steps for recording data; for more information, see "Recording data" on page 2837.

Creating and configuring an Oracle database for recording data

To record data to an Oracle database, create the database tables and configure an ODBC definition. Configure your integration server so that it can connect to the database.

Before you begin

Read the following topics:

- "Record and replay" on page 2835
- "Recording, viewing, and replaying data" on page 2834
- "Creating and configuring a database for recording data" on page 2841

About this task

Procedure

1. Use the script that is provided with IBM App Connect Enterprise to create and configure the Oracle database tables in the default table space, which is where your recorded data will be stored. Note that the script creates some tables that are not currently used, and are reserved for future use, such as WMB_EVENT_FIELDS and WMB_EVENT_TYPES.

   a) Locate the script for your operating system:

   - Windows: *install_dir*\server\ddl\oracle\DataCaptureSchema.sql
   - Linux or UNIX: *install_dir*/server/ddl/oracle/DataCaptureSchema.sql

   install_dir is the location of your IBM App Connect Enterprise installation.

   b) On Oracle, enter the following command to run the script:

```
sqlplus user/password @DataCaptureSchema.sql
```

2. Create an ODBC definition for the database.

   If you used the supplied script to create your database without modifications, create an ODBC definition for the database called MBRECORD, with MBRECORD as the data source name (DSN). For more information, see "Enabling ODBC connections to the databases" on page 184.

3. Use the **mqsisetdbparms** command to set a user identifier and password for the integration node to use when connecting to the database; for example:

   ```
   mqsisetdbparms -w workDir -n dataSourceName -u userID -p password
   ```

   - *workDir* is the name of the work directory for your integration server.
   - *dataSourceName* identifies the database to which you want to record data.
   - *userID* and *password* specify the user identifier and the password that the integration server uses to connect to the database.
4. To ensure that the changes to the **mqsisetdbparms** command take effect, restart the integration server.

   For more information, see "Starting an integration server" on page 250.
5. Test the connection to your database by using the **mqsicvp** command.

   For more information, see command.

What to do next

Continue to follow the steps for recording data; for more information, see "Recording data" on page 2837.

Creating and configuring a Microsoft SQL Server database for recording data

To record data to a Microsoft SQL Server database, create the database, and configure an ODBC definition to it. Configure your integration server so that it can connect to the database.

Before you begin

Read the following topics:

- "Record and replay" on page 2835
- "Recording, viewing, and replaying data" on page 2834
- "Creating and configuring a database for recording data" on page 2841

About this task

The following steps describe how to create the SQL Server database, create an ODBC definition, and set a user ID and password for the database.

You can use Integrated Windows Authentication to specify a Windows user account to be used for authentication when using Microsoft SQL Server to record and replay data. For more information, see "Using Integrated Windows Authentication when recording data with a Microsoft SQL Server database" on page 2845.

Procedure

1. Use the script that is provided with IBM App Connect Enterprise to create and configure an SQL Server database to store your recorded data. Note that the script creates some tables that are not currently used, and are reserved for future use, such as WMB_EVENT_FIELDS and WMB_EVENT_TYPES.
   a) Locate the script at *install_dir*`\server\ddl\sqlServer\DataCaptureSchema.sql`, where *install_dir* is the location of your IBM App Connect Enterprise installation.
   b) Optional: Customize the provided DataCaptureSchema script.

      If you modify the SQL to specify a particular schema, you must also set the same schema name in the `server.conf.yaml` file. See "Configuring data recording" on page 2839.
   c) To run the script, at a command line, navigate to the script location and enter the following command:

      ```
      sqlcmd -d databaseName -i DataCaptureSchema.sql
      ```

2. Create an ODBC definition for the database.

   If you used the supplied script to create your database without modifications, create an ODBC definition for the database that is called MBRECORD, with MBRECORD as the data source name (DSN). For more information, see "Enabling ODBC connections to the databases" on page 184.
3. Use the **mqsisetdbparms** command to set a user identifier and password for the integration server to use when it connects to the database.
   For example:

   ```
   mqsisetdbparms -w workDir -n dataSourceName -u userID -p password
   ```

   - *workDir* is the name of the work directory for your integration server.
   - *dataSourceName* identifies the database to which you want to record data.
   - *userID* and *password* specify the user identifier and the password that the integration server uses to connect to the database.

   Windows If Integrated Windows Authentication is being used for SQL Server database access, then the service user ID under which the process runs is used by Windows to access the SQL Server database. That is, it ignores any user ID and password credentials that are set using the **mqsisetdbparms** command.
4. To ensure that the changes to the **mqsisetdbparms** command take effect, restart the integration server.

   For more information, see "Starting an integration server" on page 250.
5. Test the connection to your database by using the **mqsicvp** command.

   For more information, see command.

What to do next

Follow the steps for recording data. See "Recording data" on page 2837.

Using Integrated Windows Authentication when recording data with a Microsoft SQL Server database

Integrated Windows Authentication (IWA) refers to a set of authentication protocols that are used by Windows clients and servers. You can use IWA with IBM App Connect Enterprise to provide transport-level security when you are recording data with a Microsoft SQL Server database.

Before you begin

Read the following topics:

- "Record and replay" on page 2835
- "Recording, viewing, and replaying data" on page 2834
- "Creating and configuring a database for recording data" on page 2841

About this task

You can use IWA to specify a Windows user account to be used for authentication when using Microsoft SQL Server to record and replay data. The following steps describe how to create the SQL Server database, create a Windows user account, and give that user account permission to access the SQL Server.

For more information about how to set up Microsoft SQL Server for record and replay without using IWA, see "Creating and configuring a Microsoft SQL Server database for recording data" on page 2844.

Procedure

1. Use the script that is provided with IBM App Connect Enterprise to create and configure an SQL Server database to store your recorded data. Note that the script creates some tables that are not currently used, and are reserved for future use, such as WMB_EVENT_FIELDS and WMB_EVENT_TYPES.
   a) Locate the script at *install_dir*\server\ddl\sqlServer\DataCaptureSchema.sql, where *install_dir* is the location of your IBM App Connect Enterprise installation.
   b) Optional: Customize the provided DataCaptureSchema script.
   c) To run the script, at a command line, navigate to the script location and enter the following command:

   ```
   sqlcmd -d databaseName -i DataCaptureSchema.sql
   ```

2. Create a Windows user account to be used for authentication.
   a) Ensure that you are logged on as the administrator, then create a new user account with non-administrator privileges. Provide a user name and password, ensuring that the user name has fewer than 12 characters.
   b) Add this user to the mqbrkrs group by using the **mqsisetsecurity** command.
3. Create an integration node and set the serviceUserId and servicePassword to the user name and password of the authenticating Windows user account.
4. Ensure that you are logged on as the SQL Server administrator, then give the Windows user account permission to access the SQL Server instance.
   a) Start the SQL Server Management Studio.

      For more information, see your SQL Server documentation.
   b) Connect to the database instance for which you want to set up authentication.
   c) In the instance tree, navigate to **Security** > **Logins**, right-click **Logins**, and select **New Login**.
   d) Ensure that **Windows Authentication** is selected.
   e) Click **Search**, locate the user ID to use, then click **OK**.

      The user ID typically has the format DOMAIN/USERNAME. To search the instance location, click **Advanced**, then click **Find Now**. You can then select the name from the list of possible options.
   f) To map the user ID to the databases that are used for recording and replaying data, navigate to the **User Mapping** page in the instance tree. Select the relevant databases and ensure that the correct DOMAIN/USERNAME combination and default database schema are specified.
   g) To create the user login, accept all other default values and click **OK**.
5. Create the necessary ODBC connections that target the databases that are used for recording and replaying data. This step involves defining the data source that the record and replay store uses.
   a) Open the Windows Control Panel and navigate to **System and Security** > **Administrative Tools** > **Data Sources (ODBC)**.
   b) On the **System DSN** tab, create a new data source by clicking **Add**.
   c) Select **SQL Server Native Client**, then click **Finish**.
   d) Provide a name for your ODBC data source, select the relevant SQL Server, then click **Next**.
   e) Ensure that SQL Server is set to verify authenticity by using Integrated Windows authentication, then click **Next**.
   f) Change the default database to use the database that is used as the store for recording and replaying data, then click **Next**.
   g) Accept the remaining default value, then click **Finish**.
   h) Test that the data source connects successfully to the SQL Server database by clicking **Test Data Source**.

      If the test is successful, a success message is issued. To complete setup, click **OK**.

What to do next

Next:

Create the record and replay store by following the steps in "Recording data" on page 2837. Ensure that the data source that you created in this task is used as the target for the record and replay store.

Configuring monitoring for recording

Determine which data you want to record, and configure your message flows to emit this data, by using monitoring events. Learn about the differences between how you configure event sources for record and replay and for business-level monitoring.

Before you begin

Create and configure a database, and specify the IBM App Connect Enterprise runtime properties that are required for recording data. See "Recording data" on page 2837.

About this task

Record and replay is based on a publish/subscribe model. To specify the data that you want to record to a database, you configure your message flows to emit event messages. These messages contain the data that you want to record. The messages are published to a topic that you specify when you update the `server.conf.yaml` file. See "Configuring data recording" on page 2839. The integration server subscribes to the topic and routes the published messages to the database.

The mechanism that is used for configuring message flows to emit event messages for record and replay is also used for business-level monitoring. This mechanism is summarized in the concept topic "Configuring monitoring for message flows" on page 2788, and is described further on in this topic. Because record and replay and business-level monitoring can share event sources and monitoring events, you must ensure that the configuration for one does not inadvertently affect the other. Before you begin to configure monitoring for record and replay, you need to be aware of some specific considerations that relate to the use of the monitoring mechanism for recording data for record and replay.

- When you view your recorded data in the web user interface, you might see one or more errors in the **Exception** column. You can retrieve the exception data for these errors only if you included the exception list (`$ExceptionList`) when you created your monitoring events.
- You can record messages from many different providers and sources. However, you can replay messages to IBM MQ destinations only.

  When you configure a monitoring event, you can choose to include all of the message bitstream data, the headers only, or the message body only. When an integration server replays a message, it needs to include an IBM MQ message header. It handles this requirement in one of the following ways:

  – If the recorded data includes an IBM MQ header, the integration node uses the existing IBM MQ message header.
  – If the recorded data does not include an IBM MQ header, the integration server generates an IBM MQ header. All other headers are appended to the message payload.

Procedure

To configure monitoring on a message flow to emit events for record and replay, use the following method:

- Configure and enable event sources, and activate monitoring for the message flow, by completing the steps in "Configuring monitoring for message flows" on page 2788.

The subscription to the monitoring events is durable, which results in messages being put to the queue specified in the `queueName` property of the `server.conf.yaml` file. Messages can be put to this queue even when the integration server is not running. For more information about how you can monitor the queue depth and tune data recording, see "Tuning data recording" on page 2848.

What to do next

Review the steps that you completed for recording data; see “Recording data” on page 2837. Now you are ready to view the data that you recorded; see “Viewing recorded data” on page 2849.

Tuning data recording

You can change the integration server configuration to improve performance and message throughput when you are recording data. You can configure the number of integration servers and queues used to process the data, the size of the thread pool that is used by each integration server, and how often data is committed. You can also use more than one database for storing data by creating multiple record and replay stores.

About this task

You can modify the `server.conf.yaml` file to tune recording for your integration server. See “Configuring data recording” on page 2839.
Consider each of the following factors of a record and replay store when designing your integration server configuration for record and replay.

threadPoolSize
: This property of a record and replay store determines the number of threads that are used by the recorder to process the monitoring topic subscriptions. The default value is 10.

commitCount
: The `commitCount` property of a record and replay store identifies the number of input messages that are processed on each thread before a sync point is taken. Decreasing the value of `commitCount` can improve performance.

commitIntervalSecs
: The `commitIntervalSecs` property of a record and replay store identifies the time interval at which a commit is taken when the `commitCount` property is greater than 1 but the number of messages processed has not reached the value of the `commitCount` property. Increasing the value of `commitIntervalSecs` can improve performance.

storeMode
: The `storeMode` property of a record and replay store sets the mode for the store to operate in. Valid values are `record`, `view`, and `all`. The default value is `all`, meaning that you can record, view, and replay the data. You can spread workload by specifying multiple stores with different integration servers for recording and viewing data specified for each one.

record and replay stores and record and replay sources
: The relationship between record and replay stores and record and replay sources is one-to-many. To use multiple sources for storing recorded data, you can define several sources. To increase the number of integration servers and queues used to process data for recording and viewing, you can change the ratio of sources to stores. For more information, see “Using multiple integration nodes and multiple independent integration servers for record and replay” on page 2851.

queue Name and backoutQueue
: The `queueName` property of a record and replay store specifies the queue that is used for holding data before it is recorded. The `backoutQueue` property of a record and replay store specifies the queue that is used for backing out messages that cannot be processed. You can use the default queues or you can create new queues so that the data is spread across multiple queues. The same `backoutQueue` can be specified for multiple record replay stores.

  You can configure IBM MQ to warn when queue depths are close to their maximum size. For more information, see the topics about "Event monitoring" in the IBM MQ documentation.

  You can also increase the maximum queue depth and the maximum message length for queues by using the **runmqsc** command.

  Complete the following steps:

1. At a command prompt, enter the following command, where *qmName* is the name of your queue manager:

```
runmqsc qmName
```

2. Confirm the current settings for your queue by running the following command, where *qName* is the name of the relevant queue; for example, SYSTEM.BROKER.DC.RECORD. If your queue name contains lowercase characters, you must enclose the queue name in single quotation marks.

```
dis qlocal(qName) maxdepth,maxmsgl
```

3. Enter the following command, where *qName* is the name of your queue, *newMaxDepth* is the new setting for maximum queue depth, and *newMaxMsgL* is the new setting for maximum message length:

```
alter qlocal(qName) maxdepth(newMaxDepth) maxmsgl(newMaxMsgL)
```

4. To exit the **runmqsc** environment, type end

Viewing recorded data

You can view a list of recorded messages, or details of individual messages.

Before you begin

Ensure that you have completed all the steps in "Recording data" on page 2837, "Configuring data recording" on page 2839, and "Configuring monitoring for recording" on page 2847.

About this task

You can view recorded data from the web user interface or from the administration REST API.

Procedure

To configure IBM App Connect Enterprise to view recorded data, complete the following steps:

1. Ensure your record and replay stores are configured to allow viewing. For more information, see "Configuring data recording" on page 2839.
2. Use one of the following methods to view your data:
   - View recorded data and administer your integration nodes or integration servers from the web user interface:
     a. Configure a web administration server, as described in "Configuring the IBM App Connect Enterprise web user interface" on page 87.
     b. Log on to the web user interface. For more information, see "Accessing the web user interface" on page 89.
     c. Select the **Data** tab of an integration node or integration server. All record and replay stores on the integration node or integration server that have been configured for viewing are displayed.
     d. Select the record and replay store containing the data that you want to view. Each row that is displayed represents a recorded message.
     e. You can select which columns will appear in the web user interface. The selection persists when you log out and when other users view the same record and replay store.
     f. You can download the bitstream data for a message by clicking the arrow in the **Data** column.
     g. You can create and apply a query to filter the list of messages in the record and replay store.
     h. You can show message creation times in either Coordinated Universal Time (UTC) or local time.
   - Use the administration REST API. For more information, see "Managing resources by using the administration REST API" on page 334.

Replaying data

When you have recorded data, you can replay it to a queue by using the web user interface or the administration REST API.

Before you begin

Configure your system to record data and to view the recorded data. See “Recording data” on page 2837 and “Viewing recorded data” on page 2849.

Procedure

To replay data to an IBM MQ queue, complete the following steps:

1. Ensure that your record and replay stores are configured to allow viewing. For more information, see “Configuring data recording” on page 2839.
2. Stop the integration server. See “Stopping an integration server” on page 254.
3. Open the `server.conf.yaml` configuration file for your integration server by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters, because they are invalid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
4. Update the `server.conf.yaml` file to define the IBM MQ queues to which you can replay data. The properties that you need to set are in the **RecordReplay** section of the `server.conf.yaml` configuration file.
   Copy and customize the MQDestinationTemplate section for each Record and Replay destination that you want to create. For example:

   ```
   RecordReplay:
              ...
              Destinations:
                 MQDestination:
                 endpointType: 'WMQDestination'             # The type of the target destination
    to which messages will be replayed. The default is WMQDestination, which is the only valid
    value.
                 endpoint: 'wmq:/msg/queue/[QUEUE]@[QMGR]'  #
   ```

   a) Change the heading **MQDestinationTemplate** to the name of your data destination, for example, `MQDestination`.

   b) Set the **endpointType** property to `WMQDestination`.

   c) Set the **endpoint** property to include the name of the queue and, optionally, the queue manager.

   If the queue manager is not specified (`'wmq:/msg/queue/[QUEUE]'`), the default local or remote queue manager for the integration server is used.
5. Start the integration server. See “Starting an integration server” on page 250.

6. Use one of the following methods to replay your data:
   - Use the IBM App Connect Enterprise web user interface:
     a. Configure the web user interface, as described in "Configuring the IBM App Connect Enterprise web user interface" on page 87.
     b. Log on to the web user interface. For more information, see "Accessing the web user interface" on page 89.
     c. Select the **Data** tab of the integration node or integration server. Selecting the **Data** tab displays all the record and replay stores in the integration node or integration server that have been configured for viewing.
     d. Select the record and replay store that contains the data that you want to replay. Each row that is displayed represents a recorded message.
     e. Select the messages that you want to replay. You can replay only messages that have data.
     f. Click **Mark for replay**. The selected messages are added to the replay list in the **Replay** tab; they are not replayed at this stage.
     g. Select the destination for your data. The list of available destinations corresponds to the destinations that you defined when you updated the `server.conf.yaml` file.
     h. You can now select the messages to replay.
     i. Click **Replay**. The selected messages are sent to the specified destination. Messages that are not delivered are highlighted. You can replay messages multiple times. To remove a message from the list, select the message and click **Remove**.
   - Use the administration REST API. For more information, see "Managing resources by using the administration REST API" on page 334.

Using multiple integration nodes and multiple independent integration servers for record and replay

You can have multiple integration nodes or multiple independent integration servers in your record and replay topology. If you use different integration nodes or different integration servers for deploying your message flows, and for recording data from those message flows to a database, then you must configure a publish/subscribe relationship between the integrations nodes or integration servers.

Before you begin

Read the concept topic "Record and replay" on page 2835.

About this task

You can use separate integration nodes or integration servers for deploying the message flows from which you want to record data and for viewing the data from a database. The integration node or integration server that is used for viewing and replaying data does not interact with the integration node or integration server that is used to record data to a database. However, if the integration node or integration server to which you deploy your message flows is not the recording integration node or integration server, then you must set up a publish/subscribe relationship between these integration nodes or integration servers.

Data recording is based on a publish/subscribe model. You configure monitoring on message flows that you deployed to an integration node or integration server, for example, to integration server *MONSERV*. This publishes to the topic that you specify when you configure your monitoring events. The topic identifies the source of the data that you want to record. You specify this topic when you update the `server.conf.yaml` file. see "Configuring data recording" on page 2839

You can record on any integration node or integration server but view and replay must use the same integration server.

In this scenario, *MONSERV* publishes on the monitoring topic and *RECSERV* subscribes to the topic. If *MONSERV* runs on queue manager *MONQM* and *RECSERV* runs on queue manager *RECQM*, then you need to configure a publish/subscribe relationship between *MONQM* and *RECQM*.

You can choose to create either a cluster or a hierarchical publish/subscribe relationship between the two queue managers. If you plan to add queue managers to your topology frequently, then a cluster relationship is more appropriate. See the topics on "Publish/subscribe topologies" in the IBM MQ product documentation online. This example uses a hierarchical relationship. In the example, values enclosed in single quotation marks can be replaced with your preferred values, but keep the quotation marks if you use lowercase characters. Complete the following steps:

Procedure

1. Establish a point-to-point channel connection between *MONQM* and *RECQM*.
   a) On a command line on the server that hosts *MONQM*, enter `runmqsc MONQM`
   b) Define a transmission queue for *MONQM* to use when forwarding messages to *RECQM*:

```
DEFINE QLOCAL('RECQM') USAGE(XMITQ) TRIGGER TRIGDATA('MONQM.TO.RECQM')
 INITQ(SYSTEM.CHANNEL.INITQ)
```

   c) Define a sender channel:

```
DEFINE CHANNEL('MONQM.TO.RECQM') CHLTYPE(SDR) TRPTYPE(TCP) CONNAME('RECQM.SERVER')
 XMITQ(XMIT)
  DESCR('Sender channel from MONQM to RECQM')
```

   RECQM.SERVER is the name of the server that hosts *RECQM*.
   d) Define a receiver channel:

```
DEFINE CHANNEL('RECQM.TO.MONQM') CHLTYPE(RCVR) TRPTYPE(TCP)
   DESCR('Receiver channel from RECQM to MONQM')
```

   e) Configure a listener on the queue manager:

```
DEFINE LISTENER ('LISTENER.TCP') TRPTYPE(TCP) PORT(PORT) CONTROL(QMGR)
```

   PORT is the port number on which the listener listens.
   f) Start the listener. Enter `START LISTENER('LISTENER.TCP')`
   g) Enter END to finish the **RUNMQSC** session.
   h) Now configure *RECQM*. On the server that hosts *RECQM*, enter the following commands at a command line:

```
runmqsc RECQM

DEFINE QLOCAL('MONQM') USAGE(XMITQ) TRIGGER TRIGDATA('RECQM.TO.MONQM')
 INITQ(SYSTEM.CHANNEL.INITQ)
DEFINE CHANNEL('RECQM.TO.MONQM') CHLTYPE(SDR) TRPTYPE(TCP) CONNAME('MONQM.SERVER')
 XMITQ(MONQM)
  DESCR('Sender channel from RECQM to MONQM')
DEFINE CHANNEL('MONQM.TO.RECQM') CHLTYPE(RCVR) TRPTYPE(TCP)
  DESCR('Receiver channel from MONQM to RECQM')
DEFINE LISTENER ('LISTENER.TCP') TRPTYPE(TCP) PORT(PORT) CONTROL(QMGR)
START LISTENER ('LISTENER.TCP')
END
```

   MONQM.SERVER is the name of the server that hosts *MONQM*, and *PORT* is the port number on which the listener listens.

   For more information about configuring channels for publish/subscribe, see the topics on "Publish/subscribe topologies" in the IBM MQ product documentation online.

2. Configure the queue manager publish/subscribe hierarchy.

   Make *RECQM* a child of *MONQM* by completing the following steps:

   a) Open a command line on the server that hosts *RECQM*, and enter: `runmqsc RECQM`

   b) Make *RECQM* a child of *MONQM*:

   ```
   ALTER QMGR PARENT('MONQM')
   ```

   To make *RECQM* a child of *MONQM*, the Parent queue manager name must be given as *MONQM*.

   Confirm that the operation worked by listing all publish/subscribe relationships for this queue manager:

   ```
   DIS PUBSUB TYPE(ALL)
   ```

   The output shows that *MONQM* is now the parent of *RECQM*:

   ```
   AMQ8723: Display pub/sub status details.
      QMNAME(RECQM)                           TYPE(LOCAL)
   AMQ8723: Display pub/sub status details.
      QMNAME(MONQM)                           TYPE(PARENT)
   ```

   c) Type END to end the **RUNMQSC** session for *RECQM*.

   d) Start a **RUNMQSC** session for *MONQM* and display the publish/subscribe types, as you did for *RECQM*. The output shows that *RECQM* is a child of *MONQM*.

What to do next

Consider the performance implications of your record and replay topology; see "Tuning data recording" on page 2848.

Complete the steps for recording data; see "Recording data" on page 2837.

Disabling and enabling Recording

Recording of data is enabled by default but can be disabled by modifying the `server.conf.yaml` file.

About this task

You might want to temporarily stop recording for an integration server for performance reasons, or to modify the integration server. You can disable recording for your integration server by modifying the `server.conf.yaml file`.

Procedure

Modify the `server.conf.yaml` file to disable or enable recording for your integration server by completing the following steps:

1. Stop the integration server. See "Stopping an integration server" on page 254.
2. Open the `server.conf.yaml` configuration file for your integration server by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, because they are invalid in YAML and would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
   The property that you need to set is in the **RecordReplay** section of the `server.conf.yaml` configuration file:

   ```
   RecordReplay:

       #recordReplayEnabled: 'true' # Valid values are true and false.
   ```

3. Set the **recordReplayEnabled** property to `false`.
4. Start the integration server. See "Starting an integration server" on page 250.

What to do next

You have now disabled recording for your integration server. When you are ready to start recording again, repeat the above procedure, and set the **recordReplayEnabled** property to `true` at step "3" on page 2853.

Activity Logs

Use Activity Logs as the first point of investigation when something unexpected happens in a message flow and associated external resources.

Before you begin

- Read the concept topic "Activity Log overview" on page 2854.
- Read the reference topic, "Activity Logs structure and types" on page 2858 for more information about the structure and content of Activity Logs, and about the *tags* that you can use to enrich the log data and filter its content.
- View and configure the runtime properties for Activity Logs, by following the steps described in "Viewing and setting runtime properties for Activity Logs" on page 2855.

About this task

Activity Logs provide immediate, basic information about what is happening in your message flows, and how they are interacting with external resources.

Procedure

- To view recent Activity Log information for individual message flows, use the Activity Log view in the web user interface.

  For more information, see "Viewing Activity Logs for message flows in the web user interface" on page 2856.
- If you would like to view activities over a longer period, you can use an Activity Log policy to write Activity Logs to files, to publish them to Logstash in an ELK stack, or both, as described in Activity Log policy (ActivityLog), and "Writing Activity Logs to files or to Logstash in an ELK stack" on page 2857.

Activity Log overview

Activity Logs help you to understand what your message flows are doing by providing a high-level overview of how IBM App Connect Enterprise interacts with external resources. They do not require detailed product knowledge, and can be used to enhance understanding of the overall system.

Activity Log messages are concise and avoid technical complexity, although more information is provided in the message detail. Because the log entries are short, uncomplicated, and focused on single activities, they can be quickly scanned and understood. Identifying patterns of behavior and changes in behavior can be done more easily with Activity Log messages than if you use more extensive product trace. The qualitative data about resources that are provided in Activity Logs complements the quantitative resource data that is provided by resource statistics. Because Activity Logs are enabled by default, they can be useful for troubleshooting. For example, if the Activity Log for a message flow shows a prolonged period of inactivity, review the Activity Logs for associated resource types. You might discover that a problem with an external resource is the reason why messages are no longer being processed by your message flow.

Activity Logs can be written to a circular file system. Use the Activity Log policy to set up file logging if you want continuous logging of activities over a long period. The policy has extensive filtering capabilities, based on *tags*. You can use tags to search or filter on various attributes, such as resource types or message flow names. All Activity Log messages have several tags, which are displayed in the log entries and can be used as search and filter keys. Individual resource types also have resource-specific tags that you can use to retrieve the data that you are interested in. For more information, see Activity Log policy (ActivityLog), and "Writing Activity Logs to files or to Logstash in an ELK stack" on page 2857.

Activity Logs can also be viewed in the web user interface. For more information, see "Viewing Activity Logs for message flows in the web user interface" on page 2856.

For more information about the types Activity Logs that are available in IBM App Connect Enterprise, see "Activity Logs structure and types" on page 2858.

Viewing and setting runtime properties for Activity Logs

Display and modify the runtime properties that govern the behavior of Activity Logs within the scope of a single integration server.

About this task

You can view and set properties for Activity Logs by modifying the `server.conf.yaml` configuration file for the integration server. For example, you can use these properties to change the number of Activity Log entries held in memory, or to limit log entries to error messages.

The properties that can be set are:

- activityLogEnabled

  This property takes a Boolean value that indicates whether Activity Log entries are kept in memory. The default value for this property is `true`.

- defaultLogSizePerThread

  This property takes an integer value that determines the maximum number of Activity Log entries that are written to memory before the log is recycled. The default value is `1000`. The property can have any positive integer value.

  If the thread on which the log is processed terminates (for example, because you redeploy or restart the integration server), the log entries that are held in memory are lost. When you diagnose a problem, save the current activity log before you redeploy or restart your integration server.

- minSeverityLevel

  This property takes an integer value that is used to specify the severity of messages that are written to the Activity Log.

  Only messages with a severity level higher than or equal to the specified value for this property are logged. The initial value of the property is `INFO`, which means that messages of all severities are written to the log.

  Valid values are `INFO`, `WARN`, and `ERROR`, where `ERROR` is the highest severity level and `INFO` the lowest.

  Specify a value of `WARN` to log messages of severity levels `WARN` and `ERROR`.

To display the current Activity Log configuration properties and then modify them, complete these steps:

Procedure

1. Open the `server.conf.yaml` configuration file for your integration server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters because they are invalid in YAML and would cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. Set the **`activityLogEnabled`** property to `true`.
3. Set the **`defaultLogSizePerThread`** property to the maximum number of Activity Log entries that you want to be written to memory before the log is recycled. The default value is `1000`.
4. Set the **`minSeverityLevel`** property to specify the minimum severity of messages that you want to be written to the Activity Log. The default is `INFO`.

5. Restart the integration server for the changes to take effect.

   For more information about how to start an integration server, see “Starting an integration server” on page 250.

Viewing Activity Logs for message flows in the web user interface

View Activity Log data for a message flow by using the web user interface.

Before you begin

- Read the concept topic “Activity Log overview” on page 2854.
- Start the web user interface for your integration node or integration server, as described in “Accessing the web user interface” on page 89.

About this task

Each row within a log represents an activity. For more information about what is displayed for each activity, see “Activity Logs structure and types” on page 2858.

Activity Logs are enabled by default. You can view the data in the web user interface if the message flow is deployed to an integration node or an integration server that is running.

To view Activity Logs in the web user interface:

- Browse to the message flow for which you want to view activity logs.
- Click the **Activity Log** tab.
- Click **Fetch Activity Log**.
- Optionally, customize the view by using the **Entries per page** field, or reordering the data in the columns.

The following procedure describes the process in more detail.

Procedure

To open the Activity Log view for a message flow, complete the following steps:

1. Click the **Open List of Options** icon for the required integration server. Then click Open in the menu.

   The **Open List of Options** icon appears within the tile for each integration server.
2. Click the **Open List of Options** icon for the required application, and then click Open in the menu.

   The **Open List of Options** icon appears within the tile for each deployed application.
3. Click the **Open List of Options** icon for the required message flow, and then click Open in the menu.

   The **Open List of Options** icon appears within the tile for each message flow.
4. Click the **Activity Log** tab to open the Activity Log view.

5. Click **Fetch Activity Log** to populate the Activity Log view.

   For each activity log entry, you can see the following information.

   - **Timestamp**: The date and time of the activity entry. Click the column header to change the sort order of the messages.
   - **Message**: The BIP message ID that is associated with the activity log entry.
   - **Thread ID**: The thread number that is associated with the activity log entry. Click the magnifying glass icon to see only messages with the same thread ID. To reset the activity log to show all messages, click the magnifying glass icon again.
   - **Node**: The message flow node that the activity log relates to.
   - **Resource Manager**: The resource manager (if any) that is associated with the activity log entry.
   - **Message Details**: A summary of the message. Click the message ID in the **Message** column in the same row to see the full contents of the message.
   - **Tag**: Information about the type of node the activity log relates to.

   Note: The activity log view loads only when you first click **Fetch Activity Log**. It does not reload if you leave the **Activity Log** tab and then return to it. To see the most up-to-date information, click **Refresh** in the tab header.
6. Optional: Use the menu in the **Entries per page** field to select the number of entries displayed on the page.

   The default value is `10`. Available values are `10`, `25`, `50`, `100`. `500`, and `1000`.
7. Optional: Use the menu in the **pages** field to select the required page number when the entries are displayed on more than one page.
8. Optional: Click one of the column headings that are listed in step “5” on page 2857 to reorder the data in forward or reverse order on the selected column.

   By default, the entries in the Activity Log view are ordered on the **Timestamp** column in chronological order.
9. Optional: Type a *Thread ID*, for example, `1234`, into the **Search** field to filter the entries in the Activity Log view to show only entries with the specified *Thread ID*. To revert to showing all entries, click the "x" in the **Search** field.

   If no activity logs match the specified *Thread ID*, the message "No activity logs match the current filter" appears.
10. Optional: Click the magnifying glass icon in one of the rows in the **Thread ID** column. If the magnifying glass icon displays a "+" sign before you click it, the Activity Log view changes to show only entries with the same *Thread ID* as the selected row. If the magnifying glass icon displays a "-" sign before you click it, the Activity Log view changes to show all activity log entries.
11. Optional: Click a message ID in the **Message** column to see the full details and the timestamp of the message.
12. Optional: Click **Export** to export the activity logs to a file. The file `activityLog.json` is created in the `Downloads` subdirectory of the user's home directory.

Writing Activity Logs to files or to Logstash in an ELK stack

You can define an Activity Log policy to write your Activity Logs to a file system, to a Logstash input in an ELK stack, or both. Configure the policy to capture the Activity Log data that you are interested in.

About this task

Use the filtering capabilities of this feature to customize the log content and make it relevant to your needs. You can configure the size and number of log files and the criteria by which the log rotates from one file to the next. You can also identify the location and file name of the log files.

You can use the Activity Log policy to configure the following Activity Log file characteristics:

- Whether file logging is enabled

- The maximum number of files used for each log
- The maximum size of the log files (before the log rotates to the next file)
- The maximum age of the log files (before the log rotates to the next file)
- Filters that determine the content of the logs
- The severity levels of messages logged
- The log path and file name
- Whether messages are formatted with inserts included in the message text
- The scope of the policy (which integration server it applies to)

You can also use the Activity Log policy to configure the following Activity Log ELK properties:

- Enable or disable publication to ELK
- Reference the ELk Connection configuration to be used

For more information about the properties that can be set and their possible values, see Activity Log policy (ActivityLog). The scope of the settings is the integration server. For example, if you set the **Number of log files** property to 5, each integration server has a maximum of five files for its Activity Log.

The files are in UTF-8 format.

Procedure

1. Create an Activity Log policy by using the Policy editor in the IBM App Connect Enterprise Toolkit (see "Creating policies with the IBM App Connect Enterprise Toolkit" on page 327).
2. Define appropriate values for the properties in the policy, as described in Activity Log policy (ActivityLog).

What to do next

Deploy the Activity Log policy before or with the message flows that it applies to.

Activity Logs structure and types

Activity Logs contain information about message flows and how they interact with external resources. They can be generated for a message flow or a resource type. Learn about the structure and content of Activity Logs, and about the *tags* that enrich the log data and can be used to filter its content.

You can choose to write activity logs to a file, as described in "Writing Activity Logs to files or to Logstash in an ELK stack" on page 2857.

You can also display Activity Log data in the Activity Log view of the web user interface, as described in "Viewing Activity Logs for message flows in the web user interface" on page 2856.

Activity Log columns

The data in Activity Logs is organized in columns. The following columns are displayed by default:

Table 93. Log Columns

Column title	Description
MessageNumber	The message identifier
Severity	The severity of the message (whether the message is informational, a warning or an error)
Timestamp	The date and time that the activity occurred (taken from the computer that is hosting the integration server)
ThreadID	The identifier of the thread on which the activity message was generated.

Table 93. Log Columns (continued)	
Column title	**Description**
FormattedMessage	The formatted activity message (with all inserts resolved)
Inserts	The message inserts
RM	A resource type name
MSGFLOW	A message flow name

Activity Log tags

One of the important features of the Activity Log is the use of *tags* to enrich the log data. *Tags* are strings that represent significant categories of data that can be used for filtering and for providing additional information.

In the Log Columns table, the column headings RM and MSGFLOW are examples of tags.

Some tags are generic across all types of log, but some are specific to individual resource types. The following tags apply to all types of Activity Log:

Table 94. Generic Activity Log tags.		
Name	**Description**	**Valid values**
RM	The name of the resource type to which the activity relates	• Aggregation • CICS • Connect:Direct • GlobalCache • JDEDWARDS • JMS • Kafka • Loopback • MQTT • ODBC • ODM • PEOPLESOFT • REST • Salesforce • SAP • SIEBEL
MSGFLOW	The name of the message flow to which the activity relates	Any valid message flow name
NODE	The name of the message flow node from where the Activity Log message is generated.	Any valid message flow node name

Table 94. Generic Activity Log tags. (continued)		
Name	**Description**	**Valid values**
NODETYPE	The type of the message flow node from where the Activity Log message is generated.	• INPUT • OUTPUT • READ • REQUEST • REPLY • RECEIVE
HOSTNAME	The name of the host server where the Activity Log message is generated.	Any valid hostname
PORT	The listening port of the host server where the Activity Log message is generated.	Any valid port
CONNECTION_EVENT	An event that relates to a change in the state of the connection.	• CONNECTION_EVENT.CONNECTED • CONNECTION_EVENT.PLANNED_DISCONNE • CONNECTION_EVENT.UNPLANNED_DISCON • CONNECTION_EVENT.FAILURE
FILENAME	A file name	Any valid file name
FILEPATH	The fully qualified path to the file specified using the `FILENAME` tag	Any valid file path

For more information about the types of activities that are logged for individual resource types, and about the additional tags for particular resource types, see the following topics:

- "Aggregation Activity Log" on page 2861
- "CICS Activity Log" on page 2861
- "Global Cache Activity Log" on page 2862
- "IBM Sterling Connect:Direct Activity Log" on page 2863
- "JMS Activity Log" on page 2864
- "Kafka Activity Log" on page 2865
- "LoopBack Activity Log" on page 2865
- "MQTT Activity Log" on page 2866
- "ODBC Activity Log" on page 2866
- "ODM Activity Log" on page 2867
- "REST Request Activity Log" on page 2868
- "Salesforce Activity Log" on page 2868
- "WebSphere Adapters Activity Log" on page 2869

Aggregation Activity Log

Provides a high-level overview of how IBM App Connect Enterprise processes aggregation requests with external systems to better understand which aggregations are in progress and waiting for responses from end systems.

Activity Log content

Activity Log entries for Aggregation include the following types of activity:

- An aggregation group is started.
- All the aggregation requests that have been sent to end systems.
- All the aggregation requests that have received responses from end systems.
- The successful completion of an aggregation, containing all expected replies.
- The timeout of an aggregation, containing all received replies.
- Any unknown replies received from end systems.

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to Aggregation. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see “Activity Logs structure and types” on page 2858. The following table lists the additional Aggregation tags.

Table 95. Activity Log tags.

Name	Description	Valid values
AGGREGATE_NAME	The name of the aggregation as defined on the aggregation control node.	Any valid aggregate name string.
FOLDER	The name of the aggregate request folder for a request.	Any valid aggregate folder name string.
AGGREGATE_ID	The unique id for an aggregation	A 24-byte binary field that is displayed as 48 hex characters.
REPLY_ID	The unique id for an aggregation request to an end system.	A 24-byte binary field that is displayed as 48 hex characters.
PROTOCOL	The type of transport that is used for sending the aggregate request.	Any protocol that is supported by the aggregate nodes.
REPLIES	The number of aggregate replies received.	Any valid positive integer.
TIMEOUT	The time in seconds for the aggregate timeout.	A positive decimal with only one decimal place.

CICS Activity Log

Learn about the specifics of CICS Activity Logs.

Activity Log content

Activity Log entries for CICS include the following types of activity:

- A CICS program was called successfully

- A call to a CICS program failed
- A security identity contained in a message was propagated to CICS
- A security failure occurred when calling CICS
- A communications failure occurred when connecting to a CICS server
- A new transaction was started during flow processing
- CICS resources were committed during a flow transaction
- CICS resources were rolled back during a flow transaction
- A CICS abend occurred
- A timeout occurred during a CICS request
- An SSL connection failed when connecting to a CICS server

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to CICS. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see "Activity Logs structure and types" on page 2858. The following table lists the additional CICS tags.

Table 96. Activity Log tags

Name	Description	Valid values
CICS_SERVER	The CICS system against which the activity has been performed	Any valid CICS system
CICS_PROGRAM	The initiated CICS program	Any valid CICS program

Global Cache Activity Log

The global cache activity log provides a high-level overview of how IBM App Connect Enterprise interacts with the global cache or external grid, so that you can understand what your message flows are doing.

Activity Log content

Activity Log entries for the global cache include the following types of activity:

- A message flow puts data into a map.
- A message flow gets data from a map.
- A message flow updates data in a map.
- A message flow removes data from a map.
- A message flow checks if a map contains a specified key.
- A message flow fails to complete an action on a map.
- An integration server connects successfully to an embedded cache or external grid.
- An integration server fails to connect to an embedded cache or external grid.
- A catalog server started successfully.
- A catalog server failed to start.
- A container server started successfully.
- A container server failed to start.

Activity Log tags

In addition to the generic Activity Log tags, more tags are specific to the global cache. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see "Activity Logs structure and types" on page 2858. The following table lists the additional global cache tags.

Name	Description	Valid values
CACHEMAP	The name of the map in the global cache.	Any valid map name.
CACHENAME	The name of the WebSphere eXtreme Scale grid for this cache.	For the embedded global cache, the cache name is WMB. For an external grid, this value is the grid name that is specified in the policy.

IBM Sterling Connect:Direct Activity Log

Learn about the specifics of Connect:Direct Activity Logs.

Activity Log content

Activity Log entries for Connect:Direct include the following types of activity:

- A file is sent from a CDOutput node.
- A file is received by a CDInput node.
- A node starts to process a file.
- A node finishes processing a file.

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to Connect:Direct. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see "Activity Logs structure and types" on page 2858. The following table lists the additional Connect:Direct tags.

Table 97. Activity Log tags.

Name	Description	Valid values
CDPNODE	The primary Connect:Direct server	Any valid Connect:Direct server
CDSNODE	The secondary Connect:Direct server	Any valid Connect:Direct server
CDPROCESSNAME	The name given to a process that is run in Connect:Direct	Any valid process name
CDPROCESSNUMBER	The identifier of the process specified in the *CDPROCESSNAME* tag	Any valid process number

JMS Activity Log

Learn about the specifics of JMS Activity Logs, including the types of activity logged and the tags that can be used for filtering on these activities.

Activity Log content

Here are some examples of the type of JMS node activities that are logged in a JMS Activity Log:

- Connected to a JMS provider
- Created a non-transactional session for a JMS provider
- Received a message from a queue
- Browsed a queue
- Sent a message to a queue
- Published to a topic
- Timed out, waiting for a message on a queue
- Sent a message to a backout queue
- Failed to connect to a JMS provider

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to JMS. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see “Activity Logs structure and types” on page 2858. The following table lists the additional JMS tags.

Table 98. Activity Log tags.

Name	Description	Valid values
JMSPROVIDER	The JMS provider configured on the JMS node	Any valid JMS provider
CONNECTIONFACTORYNAME	The name of the connection factory that is used to create a connection to the JMS provider	Any valid connection factory name
JNDIBINDINGSLOCATION	The location used to look up Java Naming and Directory Interface (JNDI) Administered objects such as connection factories and destinations	Any valid system path or Lightweight Directory Access Protocol (LDAP) location
INITIALCONTEXTFACTORY	The fully qualified class name of the class used to perform JNDI lookups	Any valid initial context factory
JMSTRANSACTIONMODE	The transaction mode configured on the JMS node	• yes • no

Kafka Activity Log

The Kafka Activity Log provides a high-level overview of how IBM App Connect Enterprise interacts with Apache Kafka servers by using the KafkaProducer and KafkaConsumer nodes.

Activity Log content

When a KafkaProducer node publishes a message, or a KafkaConsumer node receives a message, or a KafkaRead node reads a message, information is written to the activity log.

Activity Log messages for the KafkaProducer node contain the following information:

- Topic name
- Partition name
- Offset
- Client ID

Activity Log messages for the KafkaConsumer node contain the following information:

- Topic name
- Partition name
- Offset
- Client ID
- Group ID

Activity Log messages for the KafkaRead node contain the following information:

- Topic name
- Partition name
- Offset
- Client ID

This information enables you to review the interactions between your message flows and the Kafka servers.

LoopBack Activity Log

The LoopBack Activity Log provides a high-level overview of how IBM App Connect Enterprise interacts with LoopBack connectors by using the LoopBackRequest node.

Activity Log content

When a LoopBackRequest node issues a request for an operation through a LoopBack connector, information is written to the activity log for successful and failed processing.

The LoopBackRequest node activity log messages contain the following information:

- The name of the operation: create, retrieve, update, or delete.
- The data source name.
- The object name.
- The ID, where applicable; for example, it is provided for an operation to delete records by ID, but not for a retrieve operation without a filter.
- The status of the operation, indicating whether the operation is about to start, has completed successfully, or has failed.

Activity Log entries that are written by the LoopBackRequest node are not associated with a message flow and are not visible in the Activity Log Viewer in the web user interface; however, they are available when an Activity Log policy is used to write your activity logs to a file system (see Activity Log policy (ActivityLog)).

MQTT Activity Log

Learn about the types of activity that are logged in the MQTT Activity Log, and the tags that can be used for filtering these activities.

Activity Log content

Here are some examples of the type of MQTT node activities that are logged in an MQTT Activity Log:

- Successfully connected to an MQTT server
- Failed to connect to an MQTT server
- Reattempted the connection to an MQTT server
- Published a message to a topic
- Failed to publish a message to a topic
- Subscribed to a topic
- Received a message that was published to a topic
- Lost the connection to an MQTT server
- Successfully disconnected from an MQTT server
- Failed to disconnect from an MQTT server

Activity Log tags

In addition to the generic Activity Log tags, there are extra tags that are specific to MQTT. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see “Activity Logs structure and types” on page 2858. The following table lists the additional MQTT tags.

Table 99. Activity Log tags.

Name	Description	Valid values
TOPICNAME	The name of the topic that messages are published to, or the name of the topic that is subscribed to.	Any valid MQTT topic name. For more information, see MQTT Protocol Specification.
CONNECTIONURL	The URL of the MQTT server.	Any valid URL.

ODBC Activity Log

Provides a high-level overview of how IBM App Connect Enterprise interacts with databases so that you can better understand these interactions.

Activity Log content

Activity Log entries for ODBC include the following types of activity:

- A message flow connects to an ODBC datasource.
- A message flow prepares a database statement for execution.
- A message flow executes a database statement.

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to ODBC. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see “Activity Logs structure and types” on page 2858. The following table lists the additional ODBC tags.

Table 100. Activity Log tags.

Name	Description	Valid values
DATASOURCE	The name of the ODBC datasource which is suffixed with XA for globally coordinated connections.	Any valid ODBC datasource name.
ODBCOPERATION	The type of ODBC interaction.	CONNECT, PREPARE, and EXECUTE.

ODM Activity Log

The ODM Activity Log provides a high-level overview of the execution of Operational Decision Manager (ODM) rulesets through the ODMRules or JavaCompute node.

Activity Log content

ODM Activity Log entries are written when the following events are generated by an node or node:

- The execution of a ruleset completes successfully (BIP11200)
- The execution of a ruleset fails (BIP11201)
- A ruleset is about to be executed (BIP11203). This information enables you to see how long the ODM ruleset took to execute, by calculating the time between this event and the success or failure event.

The messages that are written to the ODM Activity Log also include the following information:

- The name of the ODM Server policy
- The ruleset path
- The number of rules matched
- The number of input parameters
- The number of output parameters

Activity Log tags

In addition to the generic Activity Log tags, there are tags that are specific to the execution of ODM business rules. These tags enrich the logged information and can also be used for filtering.

For information about the generic tags, see “Activity Logs structure and types” on page 2858. The following table lists the additional tags for the execution of ODM business rules:

Table 101. Activity Log tags.

Name	Description	Valid values
ODM	The ODM Server policy that specifies the details of either a remote Operational Decision Manager (ODM) rule execution server or a ruleset archive on the local file system.	The name of an ODM Server policy, in the form *{PolicyProject}:policyName*. For more information, see ODM Server policy (ODMServer).

Table 101. Activity Log tags. (continued)		
Name	**Description**	**Valid values**
RULESMATCHED	The number of rules that were matched in the ruleset. This tag is available only when the execution of a ruleset completes successfully (BIP11200).	Any value that is greater than or equal to 0.

For more information about using ODM rules with IBM App Connect Enterprise, see "Using Operational Decision Manager (ODM) business rules" on page 1241.

REST Request Activity Log

The REST API activity log provides a high-level overview of how IBM App Connect Enterprise interacts with REST APIs by using the RESTRequest and RESTAsyncRequest nodes.

Activity Log content

When a RESTRequest or RESTAsyncRequest node issues a request to a REST API, information is written to the activity log for successful and failed processing.

The activity log messages for the REST request nodes contain the following information:

- Operation name.
- HTTP method.
- URL used.
- HTTP status code of the response.
- Size of the request headers and body.
- Size of the response headers and body.
- Total time for the request.

  You can use the activity log to determine how much time is being spent outside of the REST request node when a request is issued to a remote REST API. The time written in the activity log includes only the time spent calling the operating system's send and receive system calls, and the time spent waiting for the remote REST API to process and respond to the request. The time written excludes the time spent elsewhere in the REST request node, such as time spent parsing or serializing message tree data. If this time is unexpectedly high, it indicates a performance problem with the network or the remote REST API rather than the REST request node. The time can be compared with the information available through message flow statistics to determine if a performance problem is occurring inside or outside of the REST request node.

Activity Log entries written by the RESTRequest and RESTAsyncRequest nodes are visible in the Activity Log Viewer in the web user interface. They are also available when an Activity Log policy is used to write your Activity logs to a file system.

Salesforce Activity Log

The Salesforce activity log provides a high-level overview of how IBM App Connect Enterprise interacts with Salesforce by using the SalesforceRequest node.

Activity Log content

When a SalesforceRequest node issues a request against Salesforce, information is written to the activity log for successful and failed processing. Activity Log entries for SalesforceRequest node operations include the following types of activity:

- A SalesforceRequest operation completed successfully.

- A SalesforceRequest operation failed.

Activity Log entries that are written by the SalesforceRequest node are not associated with a message flow and are not visible in the Activity Log Viewer in the web user interface. However, they are available when an Activity Log policy is used to write your activity logs to a file system (see Activity Log policy (ActivityLog)).

WebSphere Adapters Activity Log

Learn about the specifics of WebSphere Adapters Activity Logs.

Activity Log content

Activity Log entries for WebSphere Adapters include the following types of activity:

- An adapter connected to an Enterprise Information System (EIS).
- An adapter made an outbound request.
- An outbound request failed.
- An adapter received an inbound message.
- An adapter closed a connection.

Activity Log tags

In addition to the generic Activity Log tags, there are additional tags that are specific to WebSphere Adapters. These tags enrich the logged information and can also be used for filtering.

For more information about the generic tags, see "Activity Logs structure and types" on page 2858. The following table lists the additional WebSphere Adapters tags.

Table 102. Activity Log tags.

Name	Description	Valid values
ADAPTER	The adapter name	Any valid adapter name

Reporting logs and monitoring events to a Logstash input in an ELK stack

IBM App Connect Enterprise provides a facility to send any or all of your BIP message logs, message flow monitoring events, and activity logging events for your integration servers to a Logstash input in an Elasticsearch, Logstash, and Kibana (ELK) stack, so that you can view that data in a Kibana dashboard.

The ELK stack consists of three parts:

- **Elasticsearch** (a search and analytics engine)
- **Logstash** (a data processing pipeline that ingests data and sends it to a stash, such as Elasticsearch)
- **Kibana** (a visualization tool that can be used to display data in charts and graphs).

For more information about Elasticsearch, Logstash, and Kibana, see the Elastic stack web pages.

BIP message logs, message flow monitoring events, and activity logging events that are generated by IBM App Connect Enterprise integration servers (not integration nodes) can be sent to Logstash. However, you can enable logging at the level of an integration server or an integration node, by setting properties in the `server.conf.yaml` or `node.conf.yaml` file. To configure a specific integration server to send logs and events to Logstash, you set logging properties in the `server.conf.yaml` file for that integration server. If you enable logging for an integration node (by setting logging properties in the `node.conf.yaml` file), events that are generated by all of its managed integration servers are sent to Logstash.

App Connect Enterprise requires the ELK stack to include Logstash with either `beats` or `http` input plug-in configured. When you configure App Connect Enterprise to send to Logstash, you select the `beats`

or `http` protocol and specify connection and security parameters. When the reporting feature is active, logging and event data are sent to Logstash at regular intervals, which you can specify in the `.conf.yaml` file. You can also specify the Logstash input protocol to be used for sending the data (`beats`, `beatsTls`, `http`, or `https`). For more information about how to configure your integration servers to report logging and event data to Logstash, see "Configuring integration servers to send logs and events to Logstash in an ELK stack" on page 2870.

When you activate the IBM App Connect Enterprise logging capability and specify the Logstash input protocol, BIP message logs, message flow monitoring events, and activity logging events that are triggered by integration server events are sent to the Logstash input plug-in. The logs and events can be published from independent integration servers and from integration servers that are managed by an integration node. The logging data that is sent to Logstash contains information about the events that are issued by the integration server process. Events that are initiated by other components (such as an integration node) are not reported. If you compare the contents of the local event log relating to integration servers with the logging data that is published to Logstash, you see that a very small number of BIP message logs, message flow monitoring events (or both) at the beginning and end of the local log are not published to Logstash. These logs and events are typically messages about the startup and shutdown of the integration server. Publication of logs and events to Logstash begins after the integration server starts, which means that messages that relate to the integration server startup are not published to Logstash. The logs and events are then published until the integration server shutdown process begins, so any messages that relate to the shutdown are written to the local system log but are not published to Logstash. If an integration server does not appear to be delivering messages to Logstash, check the local system log for information. For more information about Logstash, see the Logstash reference documentation online.

- If you want to send data to a secured Logstash input, you can configure the security credentials, including the username, password, truststore, and keystore, by setting properties in the integration node's `node.conf.yaml` file or the integration server's `server.conf.yaml` file

. You can store the security credentials in the encrypted vault by using the command and the command. For more information about encrypted security credentials, see "Configuring encrypted security credentials" on page 177.

For more information about configuring an integration server, see "Configuring an integration server by modifying the server.conf.yaml file" on page 172. For more information about viewing data in a Kibana dashboard, see the Elastic stack online documentation.

Configuring integration servers to send logs and events to Logstash in an ELK stack

You can configure your integration servers to send any or all of your logging and event data, in the form of BIP message logs, message flow monitoring events, and activity logging events, to a Logstash input plug-in in an Elasticsearch, Logstash, and Kibana (ELK) stack. This configuration enables you to display the reported information in a Kibana dashboard.

Before you begin

Read the conceptual information that is provided in "Reporting logs and monitoring events to a Logstash input in an ELK stack" on page 2869.

About this task

To enable your IBM App Connect Enterprise integration servers to send logging and event information to a Logstash input in an ELK stack, you must configure the integration node or server by setting the properties in the `node.conf.yaml` or `server.conf.yaml` file.

For more information about configuring an integration node or server, see "Configuring an integration node by modifying the node.conf.yaml file" on page 118 or "Configuring an integration server by modifying the server.conf.yaml file" on page 172.

Procedure

Complete the following steps to enable your integration servers to send logging and event data to a Logstash input in an ELK stack:

1. Decide which of the following Logstash input protocols that you want to use:

   - `beats`
   - `beatsTls`
   - `http`
   - `https`

   This information determines the property values that you set in the following steps.

   For more information about the Logstash input protocols that you can use to connect IBM App Connect Enterprise to the ELK stack, see the Logstash reference documentation online.

2. Open the `node.conf.yaml` or `server.conf.yaml` configuration file for your integration node or server, by using a YAML editor.

   If you are not able to access a YAML editor, you can edit the file by using a plain text editor. However, you must ensure that you do not include any tab characters. Tab characters are not valid in YAML files and they would cause your configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.

3. Configure the integration node or server to use your chosen Logstash input protocol (`beats`, `beatsTls`, `http`, or `https`), by setting properties in the `ELKConnections` section of the `node.conf.yaml` or `server.conf.yaml` file:

```
ELKConnections:
  # Description for ELK Connections.
  # elkConnection1:
  #   elkProtocol: 'beats'                      # Logstash input protocol. Valid values are:
 'beats', 'beatsTls', 'http', or 'https'.
  #   hostname: 'myhost.domain.com'             # Hostname for the elkProtocol endpoint.
  #   port: 0                                   # Port for the elkProtocol endpoint.
  #   uploadIntervalMilliSecs: 60000            # Interval between uploading cached data,
 set in milliseconds.
  #   elkCredential: ''                         # Set an 'elk' credential alias name to
 enable basic authentication, if it is required by the Logstash input protocol.
  #   keystoreFile: '/path/to/keystore.jks'     # Set the path to the keystore to be used,
 if it is required by the Logstash input protocol.
  #   keystorePass: 'P4s5w0rd'                  # Set the password, or 'keystore' credential
 alias to the password, of the keystore.
  #   keyAlias: ''                              # Set the alias name of the private key, if
 mutual authentication is required by the Logstash input protocol.
  #   KeyPassword: ''                             # Set the password, or 'keystorekey'
 credential alias to the password, for accessing the private mutual authentication key.
  #   truststoreFile: '/path/tp/truststore.jks'  # Set the path to the truststore to be used,
 if it is required by the Logstash input protocol.
  #   truststorePass: 'P4s5w0rd'                # Set the password, or 'truststore'
 credential alias to the password, for accessing the truststore.
```

   For example, to connect to an unsecured Logstash input plug-in by using the `beats` input protocol, on `localhost` port 5044, and to upload the log data every 30 seconds, set the following properties:

```
ELKConnections:
  elkbeats:
    elkProtocol: 'beats'
    hostname: 'localhost'
    port: 5044
    uploadIntervalMilliSecs: 30000
```

4. Optional: If you are connecting to a secured Logstash input plug-in, configure the security credentials that are required to access it. For example, if you are using HTTP to connect to a Logstash input plug-

in that requires a username and password, configure the integration node or server to use an `elk` credential for basic authentication (basicAuth), by completing the following steps:

a) Update the `node.conf.yaml` or `server.conf.yaml` file to specify a credential to be used for basic authentication (basicAuth), by specifying the credential name in the **elkCredential** property; for example:

```
elkCredential: 'elk_ID'
```

b) Use the command to create the security credential that you specified in the `.conf.yaml` file (`elk_ID`), setting the username and password, and a credential type of `elk`:

```
mqsicredentials --work-dir elk_work_dir --create --credential-name elk_ID --credential-
type elk --username user1 --password passw0rd1
```

5. Optional: Enable App Connect Enterprise BIP messages to be sent to the configured ELK Connection, by setting properties in the Log section of the `.conf.yaml` file:

```
Log:
  #consoleLog: true          # Control writing BIP messages to standard out. Set to true or
 false, default is true.
  #outputFormat: 'text'      # Control the format of BIP messages written to standard out and
 file. Set to ibmjson or text, default is text.
  #eventLog: '[iib.system-work-dir]/log/[iib.system-node-label].[iib.system-server-
label].events.txt'     # Control writing BIP messages to file. Set to '' to disable, default
 is as shown.
  #eventLogFileSize: 10      # The maximum size in MB of an event log file before it is
 rotated into a new file
  #eventLogFileCount: 10     # The maximum number of event log files that should be rotated
 between.
  #elkLog: false             # Control the publication of BIP messages to an ELK
 (Elasticsearch, Logstash, Kibana) stack. Set to true or false, default is false.
  #elkConnections: ''        # Name of the ELK connection to use, for example
 'elkConnection1'
                             # Each named ELK Connection must be defined in the
 ELKConnections section below.
```

For example, to enable logs data to be sent to the ELK Connection that was configured in the previous example (`elkbeats`), set the following properties:

```
Log:
  elkLog: true
  elkConnections: 'elkbeats'
```

6. Optional: Optionally, enable IBM App Connect Enterprise message flow monitoring events to be sent to the configured ELK Connection, by setting properties in the Events.BusinessEvents.ELK section of the `.conf.yaml` file:

```
Events:
  BusinessEvents:
    ELK:
      #enabled: false              # Set true or false, default false
      #outputFormat: 'json'      # Set json, default json
      #elkConnections: ''          # Name of the ELK connection to use, for example
 'elkConnection1', must be defined in the ELKConnections section below.
```

Also, ensure that Monitoring is enabled in the Monitoring.MessageFlow section of the `.conf.yaml file`:

```
Monitoring:
  MessageFlow:
    #publicationOn: 'inactive'         # choose 1 of : active|inactive, default inactive
```

7. Optional: Deploy an Activity Log policy to the integration server, with the **elkLog** property set to `true` and the **elkConnections** property set to match the name of an ELK connection configuration specified in ELKConnections section of the `.conf.yaml` file.

8. Restart the integration server for the changes to take effect.

   The integration server's log events in the form of BIP message logs, message flow monitoring events (or both), are sent to the configured Logstash input plug-in.

   When the integration server starts delivering BIP message logs, message flow monitoring events (or both), to the configured Logstash input, a confirmation message is written to the log; for example:

```
BIP6503I: ( CHECK.IS2 ) The integration server successfully sent data to ELK connection
 ''ELKbeats''
using elkProtocol ''beats'', hostname ''localhost'' and port ''5444''.
```

   If the integration server is unable to deliver event data to the configured Logstash input, a BIP message is logged to the integration server's local event log; for example:

```
BIP3888E: ( CHECK.IS2 ) The ELK connector ''ELKbeats'' failed to send data to
 ''localhost:5444''.
Error details: ''SocketException BIP3150E: ImbBasicSocket::connectTimeout "An error occurred
 whilst performing a socket operation:
getsockopt" [::connect::select(), 10061, No connection could be made because the target
 machine actively refused it.
```

Workload management

Workload management allows system administrators to monitor and adjust the speed that messages are processed, as well as controlling the actions taken on unresponsive flows and threads.

Before you can use workload management functions, you must ensure that the publication of events has been enabled and that a pub/sub broker has been configured. For more information, see “Configuring the publication of event messages” on page 2807 and “Configuring the built-in MQTT pub/sub broker” on page 2811.

The following topics explain the various options available under workload management:

Message flow notification

A common requirement is to be able to monitor the speed at which IBM App Connect Enterprise processes messages. Workload management allows the system administrator to express a notification threshold for individual message flows deployed. An out of range notification message is produced if the notification threshold is exceeded. A back in range notification message is produced if the notification threshold later drops back into range. For more information, see “Configure a message flow to cause notification threshold messages to be published” on page 2874.

Setting the maximum rate for a message flow

The system administrator can set the maximum rate that an individual message flow can run at. The maximum rate is specified as the total number of input messages processed every second. When set, the number of input messages that are processed across the flow is measured. This measure is irrespective of the number of additional instances in use, the number of input nodes in the message flow, or the number of errors that occur. If necessary, a processing delay is introduced to keep the input message processing rate under the maximum flow rate setting. For more information, see “Setting the maximum rate for a message flow” on page 2877.

Setting additional threads to service a message flow

You can set the number of additional threads that the integration node can use to service a message flow, and specify whether the additional threads start when the flow starts.

Unresponsive message flows

Allows you to specify and monitor the maximum amount of time that any message flow is allowed to process a message for, and to specify an action to be taken if the timeout is exceeded. Additionally, manual requests can be made to stop a message flow by restarting the integration server. For more information, see Unresponsive message flows.

Workload management policy configuration

Instead of defining properties on the message flow, you can specify a workload management policy within the Integration Registry for a message flow, so that message flows can refer to the policy to find properties at run time. You can change the values of attributes for a policy on the integration node, which then affects the behavior of a message flow without the need for redeployment. The workload management policy encompasses all of the properties available under workload management in one place, including the notification threshold and maximum rate, and therefore allows for easier tuning of message flow performance.

Configure a message flow to cause notification threshold messages to be published

How to configure and monitor the message flow notification threshold.

About this task

Details how a notification threshold can be set up on a message flow to cause a notification message to be sent if the message count for messages arriving in the flow exceeds the notification threshold. A further notification message is sent if later the message count drops back below the notification threshold.

Note: Unless otherwise stated, the notification threshold is a measure of the total messages every second.

The following sections provide further information about how you can configure the message flow notification threshold and monitor message flow performance:

- "Setting the notification threshold for a message flow" on page 2874
- "Message publication when the message rate for a message flow is out of range" on page 2875
- "Message publication when the message rate for a message flow goes back into range" on page 2876

Setting the notification threshold for a message flow

Describes how to calculate and set the message flow notification threshold.

About this task

Before activating the message flow notification threshold, the system administrator must first establish what message rate is required for the message flow in question. The message rate is a measure of the total number of messages arriving each second at the message flow from all input nodes of any type. It is the total rate and not the average rate. For example: A message flow contains two input nodes that are called A and B. If input node A received messages at the rate of 50 messages a second, and input node B received messages at the rate of 70 messages a second, then the total message rate for the message flow would be 120 messages per second.

There are two ways the notification threshold can be set for a message flow:

- Directly within a BAR file.
- As one of the attributes within a workload management policy that is defined within Integration Registry.

BAR file

The notification threshold is set within the BAR file under a property called *notificationThresholdMsgsPerSec*.

The property can be set in the following ways:

- Within the BAR file through the IBM App Connect Enterprise Toolkit editor.
- Within the BAR file through the **mqsiapplybaroverride** command line.

Additionally, once the BAR file is deployed, the property can be set dynamically within the flow through the IBM Integration API. Any change to the property is picked up immediately and does not require the flow to be restarted.

Workload management policy
Create and configure a workload management policy. For more information about workload management, see "Workload management" on page 2873.

A notification threshold value of zero, or not set, will cause the message flow notification threshold to be turned off. The default state is off.

Within the same integration server, a mixture of message flows can run along side each other, some with the notification threshold set, others with the notification threshold turned off.

Message publication when the message rate for a message flow is out of range

Lists the conditions that control the publishing of the message rate is out of range message.

Before you begin

- Ensure that the publication of events is enabled, and the pub/sub broker is correctly configured. For an integration node, the default configuration enables publication of events that use the built-in MQTT pub/sub broker. If IBM MQ is installed and there is a default IBM MQ queue manager that is specified on the integration node or independent integration server, the IBM MQ pub/sub broker is used for the publication of events. For more information, see "Configuring the publication of event messages" on page 2807 and "Configuring the built-in MQTT pub/sub broker" on page 2811.

About this task

The message rate statistics are collected at a checkpoint that occurs every 20 seconds. The total message rate is calculated at this checkpoint, and if the total message rate exceeds the notification threshold, the out of range XML message is published. If the total message rate continues to stay above the notification threshold, then no further out of range messages are published.

If you enable the notification threshold you can subscribe to the following topic:

- For publications on an MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
/application_name/library_name/message_flow_name
```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
/application_name/library_name/message_flow_name
```

where *integrationNodeName* is the name of the integration node, *integrationServerName* is the name of the integration server on that integration node, *application_name* is the name of the application on that integration server, *library_name* is the name of the library on that application, and *message_flow_name* is the name of the message flow that is deployed to the library.

In the situation where the message flow is not contained in either an application or a library, the *application_name* or *library_name* parameters must be omitted along with their enclosing forward slash (/). For example:

- If the message flow is not contained in an application or a library:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
/message_flow_name
```

or

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
```

```
/message_flow_name
```

- If the message flow is contained in an application and not in a library:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
/application_name/message_flow_name
```

or

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/AboveThreshold/integrationServerName
/application_name/message_flow_name
```

Message publication when the message rate for a message flow goes back into range

Lists the conditions that control the publishing of the message rate is back in range message.

Before you begin

- Ensure that the publication of events is enabled, and the pub/sub broker is correctly configured. For an integration node, the default configuration enables publication of events that use the built-in MQTT pub/sub broker. If IBM MQ is installed and there is a default IBM MQ queue manager that is specified on the integration node or independent integration server, the IBM MQ pub/sub broker is used for the publication of events. For more information, see "Configuring the publication of event messages" on page 2807 and "Configuring the built-in MQTT pub/sub broker" on page 2811.

About this task

The message rate statistics are collected at a checkpoint that occurs every 20 seconds and the total message rate is calculated at this checkpoint. If the total message rate previously exceeded the notification threshold, and then later the total message rate drops back into range, the back in range XML message is published.

No state is stored when the message flow is stopped, restarted, or redeployed.

When a flow is terminated, the flow termination process checks to see whether the last message published was to report that the message rate exceeded the notification threshold. In this situation, the flow termination process automatically publishes a message to report that the message flow is now back in range.

If you enable the notification threshold you can subscribe to the following topic:

- For publications on an MQ pub/sub broker:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
/<applicationName>/<libraryName>/<messageFlowLabel>
```

- For publications on an MQTT pub/sub broker:

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
/<applicationName>/<libraryName>/<messageFlowLabel>
```

where *integrationNodeName* is the name of the integration node, *integrationServerName* is the name of the integration server on that integration node, *application_name* is the name of the application on that integration server, *library_name* is the name of the library on that application, and *messageFlowLabel* is the name of the message flow that is deployed to the library.

In the situation where the message flow is not contained in either an application or a library, the *application_name* or *library_name* parameters must be omitted along with their enclosing forward slash (/). For example:

- If the message flow is not contained in an application and a library:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
```

```
/<messageFlowLabel>
```

or

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
/<messageFlowLabel>
```

where *integrationNodeName* is the name of the integration node, *integrationServerName* is the name of the integration server on that integration node, and *messageFlowLabel* is the name of the message flow that is deployed to the integration server.

- If the message flow is contained in an application and not in a library:

```
$SYS/Broker/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
/<applicationName>/<messageFlowLabel>
```

or

```
IBM/IntegrationBus/integrationNodeName/WorkloadManagement/BelowThreshold/integrationServerName
/<applicationName>/<messageFlowLabel>
```

where *integrationNodeName* is the name of the integration node, *integrationServerName* is the name of the integration server on that integration node, *application_name* is the name of the application on that integration server, and *messageFlowLabel* is the name of the message flow that is deployed to the application.

Setting the maximum rate for a message flow

Allows system administrators to restrict the maximum rate at which a message flow can run at by setting the maximum rate property.

About this task

The system administrator is able to restrict the rate that an individual message flow can run at by setting the maximum rate property. The maximum rate is specified as the total number of input messages processed every second by all the threads that are running in the message flow.

To calculate the number of threads within a specific message flow:

1. Count the number of input nodes within the flow.
2. Add on the number of additional instances that are specified for each input node.

There are two ways the maximum rate can be set for a message flow:

- Directly within a BAR file.
- As one of the attributes within a workload management policy that is defined within Integration Registry.

BAR file

The maximum rate is set within the BAR file under a property called *maximumRateMsgsPerSec*.

The property can be set in the following ways:

- Within the BAR file through the IBM App Connect Enterprise Toolkit editor.

  Note: You must refresh the content of a migrated BAR file before you can see and configure the *maximumRateMsgsPerSec* property.

- Within the BAR file through the **mqsiapplybaroverride** command line.

  Note: For example, to set the *maximumRateMsgsPerSec* property for a message flow included in an application, you can use the following sample code:

  ```
  mqsiapplybaroverride -b BARfile -k applicationName -m
   sampleFlow#maximumRateMsgsPerSec=100
  ```

  For more information, see command.

Additionally, once the BAR file is deployed, the property can be set dynamically within the flow through the IBM Integration API. Any change to the property is picked up immediately and does not require the flow to be restarted.

Workload management policy
Create and configure a workload management policy. For more information about workload management, see “Workload management” on page 2873.

A maximum rate value of zero, or not set, causes the message flow maximum rate to be turned off. The default state is off.

Within the same integration server, a mixture of message flows can run along side each other, some with the maximum rate set, others with the maximum rate turned off.

Chapter 10. Troubleshooting and support

If you are having problems with your message flow applications, use the techniques described in this section to help you to diagnose and solve the problems.

About this task

This section contains information about the various techniques that you can use to diagnose problems with IBM App Connect Enterprise.

Procedure

- "Making initial checks" on page 2880
- "Using logs" on page 3023
- "Using trace" on page 3026
- "Using dumps and abend files" on page 3039
- "Contacting your IBM Support Center" on page 3040

What to do next

You can also read the general troubleshooting guidance in the following topics:

- "Troubleshooting overview" on page 2879
- "Searching knowledge bases" on page 3042
- "Getting product fixes" on page 3042
- "Contacting IBM Software Support" on page 3043
- "Collecting diagnostics" on page 3041

For information that is specific to debugging message flows, see "Testing and troubleshooting message flows" on page 647.

Troubleshooting overview

Troubleshooting is the process of finding and eliminating the cause of a problem. Whenever you have a problem with your IBM software, the troubleshooting process begins as soon as you ask yourself "what happened?"

A basic troubleshooting strategy at a high level involves:

1. "Recording the symptoms of the problem" on page 2879
2. "Re-creating the problem" on page 2880
3. "Eliminating possible causes" on page 2880

Recording the symptoms of the problem

Depending on the type of problem that you have, whether it be with your application, your server, or your tools, you might receive a message that indicates that something is wrong. Always record the error message that you see. As simple as this sounds, error messages sometimes contain codes that might make more sense as you investigate your problem further. You might also receive multiple error messages that look similar but have subtle differences. By recording the details of each one, you can learn more about where your problem exists.

Sources of error messages:

- Problems view

- Local error log
- Eclipse log
- Activity Log
- User trace
- Service trace
- Error dialog boxes
- **mqsiexplain** command

Re-creating the problem

Think back to what steps you were doing that led to the problem. Try those steps again to see if you can easily re-create the problem. If you have a consistently repeatable test case, it is easier to determine what solutions are necessary.

- How did you first notice the problem?
- Did you do anything different that made you notice the problem?
- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If this process worked before, what has changed? (The change can refer to any type of change that is made to the system, ranging from adding new hardware or software, to reconfiguring existing software.)
- What was the first symptom of the problem that you witnessed? Were there other symptoms occurring around the same time?
- Does the same problem occur elsewhere? Is only one machine experiencing the problem or are multiple machines experiencing the same problem?
- What messages are being generated that could indicate what the problem is?

You can find more information about these types of question in “Making initial checks” on page 2880.

Eliminating possible causes

Narrow the scope of your problem by eliminating components that are not causing the problem. By using a process of elimination, you can simplify your problem and avoid wasting time in areas that are not responsible. Consult the information in this product and other available resources to help you with your elimination process.

- Has anyone else experienced this problem? See: “Searching knowledge bases” on page 3042.
- Is there a fix you can download? See: “Getting product fixes” on page 3042.

Making initial checks

Before you start problem determination in detail, consider whether there is an obvious cause of the problem, or an area of investigation that is likely to give useful results. This approach to diagnosis can often save a lot of work by highlighting a simple error, or by narrowing down the range of possibilities.

About this task

This section contains a list of questions to consider. As you go through the list, make a note of anything that might be relevant to the problem. Even if your observations do not suggest a cause straight away, they might be useful later if you have to carry out a systematic problem determination exercise.

Procedure

- “Did you log off Windows while IBM App Connect Enterprise components were active?” on page 2881
- “Are the Linux and UNIX environment variables set correctly?” on page 2881
- “Are there any error messages or return codes that explain the problem?” on page 2882

Did you log off Windows while IBM App Connect Enterprise components were active?

How to avoid problems caused by logging off while components are active.

About this task

Windows On Windows, logging off when an IBM App Connect Enterprise component (an integration node) is active can cause a problem.

You might see messages, including BIP2070, BIP2642, BIP1102, and BIP1103, in the Windows Event log.

When you log off, any queue manager that supports the IBM App Connect Enterprise component is stopped unless it is defined to run as a Windows service. The component runs Windows services and remains active, but it finds that the queue manager and queue manager objects that are associated with it are no longer available.

Procedure

To avoid this problem, set the queue manager that is used by the integration node to run as a Windows service, so that logging off Windows from does not cause this problem.

Are the Linux and UNIX environment variables set correctly?

Use the **mqsiprofile** command to set a command environment.

About this task

UNIX On Linux and UNIX systems, the basic settings are made by the **mqsiprofile** command, which is located in the following directory:

```
install_dir/bin
```

What to do next

See “Setting up a command environment” on page 64 for information about setting up the command environment.

Are there any error messages or return codes that explain the problem?

You can find details of error messages and return codes in several places.

Procedure

- **BIP messages and reason codes**

  IBM App Connect Enterprise error messages have the prefix BIP. If you receive any messages with this prefix (for example, in the UNIX, Linux, or z/OS syslog), you can search for them in the product documentation for an explanation. You can also view the full content of a BIP message by using the **mqsiexplain** command. For more information, see command.

  Windows In the Windows Event log, references to BIP messages are identified by the source name "WebSphere Integration node v6000", where v6000 can be replaced by a number representing the exact service level of your installation, for example 6001.

  IBM App Connect Enterprise messages that have a mixture of identifiers such as BIPmsgs, BIPv700, BIPv610, BIPv500, WMQIv210, MQSIv202, and MQSIv201 indicate a mixed installation, which does not work properly.

- **Other messages**

  For messages with a different prefix, such as AMQ or CSQ for IBM MQ, or SQL for DB2, see the appropriate messages and codes documentation for a suggested course of action to help resolve the problem.

  Messages that are associated with the startup of IBM App Connect Enterprise, or were issued while the system was running before the error occurred, might indicate a system problem that prevented your application from running successfully.

  A large or complex IBM App Connect Enterprise integration node environment might require some additional configuration of the environment beyond what is recommended in “Installing and uninstalling complementary products” on page 96. The need for such configuration changes is typically indicated by warning or error messages that are logged by the various components, including IBM MQ, the databases, and the operating system. These messages are normally accompanied by a suggested user response.

Can you see all of your files and folders?

How to show all files in Windows Explorer:

About this task

If you are using Windows Explorer to view your files and you cannot see all of your files and folders, such as the integration node workpath directory, this is because Windows Explorer, by default, hides some files and folders.

Procedure

1. Click **Tools** > **Folder options**. The **Folder Options** dialog box opens.
2. Click the **View** tab and select **Show hidden files and folders**.

Can you reproduce the problem?

If you can reproduce the problem, consider the conditions under which you can do so.

Procedure

- **Is the problem caused by a particular message flow?**

  If so, use the debugging facility of the IBM App Connect Enterprise Toolkit and user tracing to identify the problem.
- **Is the problem caused by a command?**

  On distributed operating systems, you issue commands at the system command line.
- **Does a problem command work if it is entered by another user ID?**

  If the command works when it is entered by another user ID, check the environment of each user. Paths, especially shared library paths, might be different. On Windows, UNIX systems, and Linux verify that all users have set up their command environment correctly; refer to sample profile for more information.

  Windows On Windows, the environment for the integration node is determined by the system settings, not by a particular user's variables. However, the user's variables affect non-integration node commands.

  UNIX On LinuxUNIX systems, only the service ID that is specified when the integration node was created can start an integration node.

  Windows On Windows, any authorized user can start an integration node.

Has the message flow run successfully before?

Sometimes a problem appears in a message flow that has previously run successfully.

About this task

To identify the cause of the problem, answer the following questions:

Procedure

- **Have you made any changes to the message flow since it last ran successfully?**

  If so, it is likely that the error exists somewhere in the new or modified part of the flow. Examine the changes and see if you can find an obvious reason for the problem.
- **Have you used all the functions of the message flow before?**

  Did the problem occur when you used part of the message flow that had never been invoked before? If so, it is likely that the error exists in that part. Try to find out what the message flow was doing when it failed by using user tracing, trace nodes, and the debugger function of the IBM App Connect Enterprise Toolkit.

  If you have run a message flow successfully on many previous occasions, check the current queue status and the files that were being processed when the error occurred. It is possible that they contain some unusual data value that invokes a rarely-used path in the message flow.

- **Does the message flow check all return codes?**

  Has your system been changed, perhaps in a minor way, but your message flow does not check the return codes it receives as a result of the change? For example:

  – Does your message flow assume that the queues that it accesses can be shared? If a queue has been redefined as exclusive, can your message flow deal with return codes indicating that it can no longer access that queue?
  – Have any security profiles been changed? A message flow could fail because of a security violation.
- **Does the message flow expect particular message formats?**

  If a message with an unexpected message format has been put onto a queue (for example, a message from a queue manager on a different operating system) it might require data conversion or a different form of processing. Also, check whether you have changed any of the message formats that are used.
- **Does the message flow run on other IBM App Connect Enterprise systems?**

  Is there something different about the way that your system is set up that is causing the problem? For example, have the queues been defined with the same maximum message length or priority? Are there differences in the databases used, or their setup?
- **Are you using any user-defined extensions?**

  There might be translation or compilation problems with loadable implementation library (LIL) files. Before you look at the code, examine the output from the translator, the compiler or assembler, and the linkage editor, to see if any errors have been reported. Fix any errors to make the user-defined extension work.

  If the documentation shows that each of these steps was completed without error, consider the coding logic of the message flow, message set, or user-defined extension. Do the symptoms of the problem indicate which function is failing and, therefore, which piece of code is in error? See "User-defined extensions overview" on page 2321 for more information.
- **Can you see errors from IBM App Connect Enterprise or external resources, such as databases?**

  Your message flow might be losing errors because of incorrect use of the failure terminals on built-in nodes. If you use the failure terminals, make sure that you handle errors adequately. See "Handling errors in message flows" on page 1883 for more information about failure terminals.

Have you made any changes since the last successful run?

Have you changed IBM App Connect Enterprise, any related software, or any hardware?

About this task

When you are considering changes that might have been made recently, think not only about IBM App Connect Enterprise, but also about the other programs with which it interfaces, the hardware, device drivers, and any new applications.

Procedure

- **Have you changed your initialization procedure?**

  On z/OS, have you changed any data sets or library definitions? Has the operating system been initialized with different parameters? Check for error messages generated during initialization.
- **Has the profile of the user who is running the commands on Linux or UNIX systems been changed?**

  If so, this might mean that the user no longer has access to the required objects and commands.
- **Has any of the software on your system been upgraded to a later release?**

  Check that the upgrade completed successfully, and whether the new software is compatible with IBM App Connect Enterprise (check the product readme.html file).

- **Do your message flows deal with the errors and return codes that they might get as a result of any changes that you have made?**

  Check that your message flow can handle all possible error situations.

Is there a problem with descriptive text for a command?

On Linux and UNIX systems, be careful when including special characters (backslash (\) and double quotation mark (") characters) in descriptive text for some commands.

About this task

If you use either of these characters in descriptive text, precede them with the escape character backslash (\) ; that is, enter \\ or \" if you want \ or " in your text.

Is there a problem with a database?

If you have database problems, complete a set of initial checks to identify errors.

Procedure

- Check that the database is started.
- Check that you have correctly completed ODBC configuration:
  - Linux UNIX On Linux and UNIX systems, check that you have created a copy of the sample ODBC configuration file (`odbc.ini`), and have modified them for your environment, and that you have not added any extra unsupported parameters.
  - Windows On Windows systems, click **Start** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to configure the connections you require.
- Check that you have correctly completed JDBC configuration.
- Check the number of database connections that are in use on DB2 for AIX. If you use local mode connections, a maximum of 10 is supported.
- If messages that indicate that `imbdfdb2v6.lil` failed to load, check that you have installed a supported database.

  Details of supported databases are given in IBM App Connect Enterprise system requirements.

Is there a problem with the network?

IBM App Connect Enterprise uses IBM MQ for inter-component communication. If components are on separate queue managers, they are connected by message channels.

About this task

Communication problems can arise between any of these resources:

Procedure

- integration nodes
- The IBM App Connect Enterprise Toolkit

What to do next

If any two components are on different queue managers, make sure that the channels between them are working. Use the IBM MQ **`display chstatus`** command to see if messages are flowing.

Use the **`ping`** command to check that the remote computers are connected to the network, or if you suspect that the problem might be with the network itself. For example, use the command **`ping`**

integrationNodeName, where *integrationNodeName* is a computer name. If you get a reply, the computer is connected. If you don't get a reply, ask your network administrator to investigate the problem. Further evidence of network problems might be messages building up on the transmission queues.

Does the problem affect all users?

On distributed systems, problems can be caused by different users having different environments.

About this task

Check whether there is a different user ID in an incoming message, or a different user ID issuing a command (or acting as the integration node's service ID).

Have you recently changed a password?

Check that you have updated passwords correctly.

About this task

If you have changed the operating system password for one of the following user IDs, the integration server or the deployed message flows might have access problems:

- The ID that you have specified for the **serviceUserId** of the integration server on Windows
- The ID that you have specified for access to a database

If these problems occur, complete one or more of the following steps:

Procedure

- Run the **mqsisetdbparms** command to redefine the correct values for the user ID and password.
- Run the **mqsireportdbparms** command to see the user ID that is defined. You can also run **mqsireportdbparms** using the **-n** and **-p** parameters and a possible password. If the entered password is correct, then the command returns information, including if the password was changed since the integration server was last started.
- Run **mqsireportdbparms** and check what user IDs are defined for the systems that require one. It might be that a security identity was not defined for the ID.
- Ensure that your passwords do not contain reserved keywords.
  For example, "IBM" is a reserved keyword in DB2.

Have you applied any service updates?

Check what service updates you have applied to your software.

About this task

If you have applied an APAR or a PTF to IBM MQ for z/OS, or an interim fix has been applied to a distributed operating system, check that no error message was produced. If the installation was successful, check with the IBM Support Center for any known error with the service update.

Procedure

- If a service update has been applied to any other product, consider the effect that it might have on IBM App Connect Enterprise.
- Ensure that you have followed any instructions in the service update that affect your system. For example, you might need to redefine a resource, or stop and restart a component.

- If you are not sure whether a service update has been applied to your system, search for and view the release notes stored in the product installation directory.

  These notes include the service level and details of the maintenance that you have applied.

Do you have a component that is running slowly?

If a particular component, or the system in general, is running slowly, you can take some actions to improve the performance.

About this task

Consider the following actions:

Procedure

- Check whether tracing is on.

  You might have started IBM App Connect Enterprise user tracing or service tracing, ODBC tracing, IBM MQ tracing, or native database tracing. If one or more of these traces are active, turn them off.
- Clear out all old abend files from your errors directory.

  If you do not clear the directory of unwanted files, you might find that your system performance degrades because significant space is used up.
- Increase your system memory.

Dealing with problems

Learn how to resolve some of the typical problems that can occur.

Before you begin

Make initial checks, see “Making initial checks” on page 2880, and check the following locations to see if you can rectify the problem:

- Logs, see “Using logs” on page 3023
- Trace, see “Using trace” on page 3026
- Dumps, see “Using dumps and abend files” on page 3039

About this task

This section contains the following topics:

Procedure

Resolving problems when you install IBM App Connect Enterprise

Work through the advice that is provided to help you to deal with problems that can arise when the product is installed on a distributed platform, such as on a Windows, or Linux system.

About this task

The following list describes problems that you might experience when you install IBM App Connect Enterprise on a distributed platform, with a corresponding solution or workaround.

You are prompted for the location of files during installation

Procedure

- **Scenario:** During the installation of IBM App Connect Enterprise, you are prompted to provide the location of certain installation files.
- **Explanation:** It is likely that your installation package is truncated and that files are missing.
- **Solution:** Check the size of the installation package and download another copy of the package if it is not the expected size.

The installation fails

Procedure

- **Scenario:** You have problems during the installation of IBM App Connect Enterprise.

- **Solution:** If the installation fails, you should receive an error message with an error code and a link to the log file where the error is explained. If you cannot identify the problem and need to contact your IBM Support Center for assistance, the following are the log files that you should send to your IBM Support contact:
  - `%LOCALAPPDATA%/Temp/IBM_App_Connect_Enterprise_11.0.0.n_`*`timestamp`*`.log`
  - `%LOCALAPPDATA%/Temp/IBM_App_Connect_Enterprise_11.0.0.n_`*`timestamp`*`_1_IIBCommonInstall.log`
  - `%LOCALAPPDATA%/Temp/IBM_App_Connect_Enterprise_11.0.0.n_`*`timestamp`*`_2_IIBServerInstall.log`
  - `%LOCALAPPDATA%/Temp/IBM_App_Connect_Enterprise_11.0.0.n_`*`timestamp`*`_3_IIBToolsInstall.log`

  where *timestamp* is the date and time of the installation.

The installation process is interrupted

Procedure

- **Scenario:** You are installing IBM App Connect Enterprise but the process is interrupted before the installation completes.
- **Explanation:** The process might be interrupted because of a problem, such as a power failure.
- **Solution:** Restart the installation wizard. If the installation was interrupted by a power failure, the installation wizard automatically restarts when you restart the computer. The installation process continues from the point where it was interrupted, and program files that are already installed are not reinstalled. Do not change the location of the installation directory when you restart the installation, otherwise the program files are split across the two directories, and IBM App Connect Enterprise will not work correctly.

Firewall software reports attempted internet access from an unexpected program

Procedure

- **Scenario:** During installation of the IBM App Connect Enterprise, your firewall software reports attempted internet access from an unexpected program.
- **Explanation:** Internet access is not required to complete the installation of the IBM App Connect Enterprise. However, the product might install an updated Microsoft C runtime library or updated version of the Microsoft.NET framework. These components might attempt internet access during the installation, which might cause firewall software to report a warning.
- **Solution:** This is normal behavior and the installation completes successfully whether you allow access or not.

`MSVCP120.dll missing` message is issued

Procedure

- **Scenario:** During installation of IBM App Connect Enterprise, a pop-up window is seen that states `mqsisetsecurity.exe has stopped working`. Clicking the details in the pop-up window shows that the MSVCP120.dll file is missing.
- **Solution:** Download and install the Microsoft Visual C++ 2010 Redistributable Package.

You experience display problems when you start the IBM App Connect Enterprise Toolkit

Procedure

- **Scenario:** You unpacked IBM App Connect Enterprise, started the IBM App Connect Enterprise Toolkit, and see one of two commonly reported errors.
- **Explanation:** Both errors typically occur if you log in remotely, or you switch user ID.
- **Solution:** Address the appropriate error with the following corresponding solution:

  `Can't open display localhost:1.0`
  Check that the DISPLAY variable is set to the correct value. If you are logged in locally, the typical value is `:0.0` or `localhost:0.0`.

  `Connection to ":0.0" refused by server`
  Run the following command, where *user* is the user ID you are logged in as:

  ```
  xauth merge ~user/.Xauthority
  ```

  If you are unable to correct these errors, contact your systems administrator for further help.

Resolving problems when you uninstall IBM App Connect Enterprise

Work through the advice provided to help you to deal with problems that can arise when the product is uninstalled.

About this task

The following list describes problems that you might experience when you uninstall IBM App Connect Enterprise on Windows, with a corresponding solution or workaround.

- On Windows: "The uninstallation process is interrupted" on page 2890

The uninstallation process is interrupted

Procedure

- **Scenario:** You are uninstalling IBM App Connect Enterprise but the process is interrupted before completion.
- **Explanation:** The process might be interrupted because of a problem, such as a power failure.
- **Solution:** Restart the uninstallation wizard. If the uninstallation was interrupted by a power failure, the uninstallation wizard automatically restarts when you restart the computer.

Resolving problems when running commands

Use the advice that is given here to help you to resolve common problems that can arise when you run commands.

About this task

- "MSVCP120.dll missing message is issued" on page 2891
- "Backing up or restoring the integration node returns error BIP1253 or BIP1262" on page 2891
- "ReportEvent() error message is issued on Windows when you attempt to run a command" on page 2891
- "A timeout error is issued when you attempt to run a command on AIX when Stack Execution Disable (SED) is enabled" on page 2892

- "A correct URL and password returns error BIP1939 when you attempt to connect to a remote host name" on page 2892
- "Running the mqsichangeproperties or mqsireportproperties commands on a resource manager returns error BIP2212" on page 2892

Windows MSVCP120.dll missing message is issued

Procedure

- **Scenario:** You are running commands and a pop-up window is seen that states `mqsisetsecurity.exe has stopped working`. Clicking the details in the pop-up window shows that the MSVCP120.dll file is missing.
- **Solution:** Download and install the Microsoft Visual C++ 2010 Redistributable Package.

Backing up or restoring the integration node returns error BIP1253 or BIP1262

Procedure

- **Scenario:** An error message is generated when you back up or restore an integration node:
  - The following message is returned when you run the **mqsibackupbroker** command:

    ```
    BIP1253E Failed to backup the integration node 'integrationNodeName' (error 'error_code')
    ```

  - The following message is returned when you run the **mqsirestorebroker** command:

    ```
    BIP1262E Could not delete the existing configuration of integration node
     'integrationNodeName'.
    ```

- **Explanation:** Both backup and restore commands require read/write access to the files that contain integration node configuration data. Access has failed and the command cannot complete its operation successfully.
- **Solution:** Typically, you might see one of these errors if you are running the backup or restore command against an active integration node, and a conflict of access has occurred.

  If your integration node is active, use the **mqsistop** command to stop the integration node, and try the operation again.

  You might also see one of these errors if the directory to which the command is writing, or from which it is reading, is temporarily inaccessible because of communications or authorization problems. Check that your user ID has access to the directory that you specified in the command.

ReportEvent() error message is issued on Windows when you attempt to run a command

Procedure

- **Scenario:** A ReportEvent () message is generated whenever you attempt to run any `mqsi*` command, The ReportEvent () message is followed by the result of the command itself.
- **Explanation:** The Windows Application Log has become full.
- **Solution:** Clear the Windows Application Log from the Event Viewer.

A timeout error is issued when you attempt to run a command on AIX when Stack Execution Disable (SED) is enabled

Procedure

- **Scenario:** You want to run an `mqsi*` command on AIX with Stack Execution Disable enabled, but the command times out even if the **-w** option is set to 360.
- **Explanation:** `mqsi*` commands attempt to create a JVM with Java Native Interface (JNI) calls. Because of Stack Execution Disable, the JVM creation fails.

  For more information about Stack Execution Disable, see IBM SDK, Java Technology Edition, product documentation online.
- **Solution:** Use the **sedmgr** command to create exemption records from Stack Execution Disable. For example, for **mqsilist**:

  ```
  sedmgr c exempt install_dir/server/bin/mqsilist
  ```

  The following commands might be affected by this issue:

  - **biphttplistener**
  - **bipbroker**
  - **bipservice**
  - **mqsideploy**
  - **mqsilist**
  - **mqsimode**
  - **mqsireloadsecurity**
  - **mqsireportresourcestats**
  - **mqsiwebuseradmin**

A correct URL and password returns error BIP1939 when you attempt to connect to a remote host name

Procedure

- **Scenario:** A BIP1939 message is generated whenever you attempt to run an `mqsi*` command where a URL and password were specified on the **-i** parameter to connect to a remote host name. The user name and password are correct, and the password contains non-alphanumeric characters.
- **Explanation:** The entered password contains URI Reserved characters. You cannot use URI Reserved characters within the URL.
- **Solution:** Convert URI Reserved characters to their equivalent percent-encoded format, and run the command again.
  For example, `@` becomes `%40`.

Running the mqsichangeproperties or mqsireportproperties commands on a resource manager returns error BIP2212

Procedure

- **Scenario:** A BIP2212 message is generated whenever you attempt to run an **mqsichangeproperties** or **mqsireportproperties** command for a valid resource manager.

- **Explanation:** The specified resource manager requires IBM MQ, but it is not available.

  There are a number of features in IBM App Connect Enterprise that require an IBM MQ installation, and for a queue manager to be associated with the integration node. For more information, see “Enhanced flexibility in interactions with IBM MQ ” on page 26.
- **Solution:** If you have not already done so, install IBM MQ. Create a new integration node, and specify a queue manager to associate with the new integration node by using the **mqsicreatebroker** command.

Resolving problems when creating resources

Use the advice given here to help you to resolve common problems that can occur when you create resources.

About this task

Look for the problem that you have encountered, and follow the guidance provided to recover from the error.

Problems when creating an integration node:

- “Message BIP8081 is issued when creating an integration node” on page 2893
- “Message BIP8246 is issued when creating an integration node that is configured to start as an IBM MQ service” on page 2893
- “Message BIP8087 is issued when creating an integration node on Windows” on page 2894
- “You cannot create files when creating an integration node on AIX” on page 2894

Problems when creating other resources:

- “Error message BIP2624 is issued when creating an integration server” on page 2894

Message BIP8081 is issued when creating an integration node

Procedure

- **Scenario:** Message BIP8081E is displayed when you are creating an integration node, the inserted message does not format correctly, and the integration node is not created.
- **Explanation:** This problem occurs because you are not a member of the correct group.
- **Solution:** Read the explanation of message BIP8081, and ask your IBM App Connect Enterprise administrator to give your user ID access to the mqbrkrs group.

Message BIP8246 is issued when creating an integration node that is configured to start as an IBM MQ service

Procedure

- **Scenario:** Message BIP8246E is displayed when you are creating an integration node that is configured to start and stop with the queue manager that is associated with the integration node.
- **Explanation:** You must be a member of the mqm group to run the **mqsicreatebroker** command with the **-d** parameter.
- **Solution:** Ask your IBM App Connect Enterprise administrator to give your user ID access to the mqm group.

Message BIP8087 is issued when creating an integration node on Windows

Procedure

- **Scenario:** When you create an integration node on Windows that has the same name as an integration node that you recently deleted, the following message might be displayed:

  ```
  BIP8087E integrationNode already exists and cannot be created.
  ```

- **Explanation:** The Windows service that is associated with the integration node that you deleted might be in a "Pending deletion" state. You cannot create an integration node with the same name, until the deletion of the Windows service has completed. The "Pending deletion" state can occur if a Windows system tool has a handle on the service.
- **Solution:** Close the system tool that has a handle on the Windows service and the service should be deleted.

You cannot create files when creating an integration node on AIX

Procedure

- **Scenario:** When you run the **mqsicreatebroker** command on IBM App Connect Enterprise for AIX, the following message is displayed:

  ```
  BIP8135E Unable to create files. Operating System return code 1
  ```

- **Explanation:** The user ID that you create for IBM App Connect Enterprise testing must have a primary group of mqbrkrs. The following example shows an AIX SMIT panel listing the **Change / Show Characteristics of a User**:

  ```
                    Change / Show Characteristics of a User

  Type or select values in entry fields.
  Press Enter AFTER making all desired changes.
  [TOP]                                  [Entry Fields]
  * User NAME                            peterc
    User ID                              [202]                    #
    ADMINISTRATIVE USER?                 false                    +
    Primary GROUP                        [mqbrkrs]                +
    Group SET                            [mqbrkrs,mqm,system,sys> +
  ```

Error message BIP2624 is issued when creating an integration server

Procedure

- **Scenario:** When you create an integration server, you get several BIP2624 messages (MQRC=2012 (MQRC_ENVIRONMENT_ERROR)), and no IBM MQ messages are processed.
- **Explanation:** You have created the integration node to run as an IBM MQ trusted application (that is, the integration node runs in the same process as the IBM MQ queue manager), but the user ID that you specified does not have the required authority.
- **Solution:** If you request the trusted application option on the **mqsicreatebroker** command by specifying the **-t** parameter, perform the appropriate steps for your operating system:

  Windows **Windows**

  Using the **-i** parameter on the **mqsicreatebroker** command, specify a service user ID that is a member of IBM MQ group mqm.

  Linux UNIX **Linux and UNIX systems**

  Specify the user ID mqm on the **-i** parameter on the **mqsicreatebroker** command.

Resolving problems when renaming resources

Use the advice in this topic to resolve common problems that can occur when you rename resources.

You rename a library that is referenced by an application or another library, but the containing project does not appear to contain the renamed library

Procedure

- **Scenario:** An application or library refers to a library. You rename that library, but the containing application or library now contains no reference to the original or renamed library in the Application Development view.
- **Explanation:** When you rename a library that is referenced by an application or library, the library reference in the containing application or library is not updated automatically.
- **Solution:** To refer to the renamed library, complete the following steps.
  a) Right-click the containing application or library and click **Manage library references**.
  The **Manage library references** dialog box opens.
  b) Select the renamed library and clear the original library, then click **OK**.
  The containing application or library now contains a reference to the renamed library. For more information, see “Referencing resources in other libraries” on page 1976.

Resolving problems when starting an integration node

Use the advice given here to help you to resolve common problems that can arise when you start an integration node.

About this task

When you start an integration node by using the **mqsistart** command, the **mqsicvp** command is run automatically to check that the environment is set up correctly (for example, the installed level of Java is supported). On Linux and UNIX, the **mqsicvp** command also verifies that the ODBC environment (if specified) is configured correctly. For more information, see command.

For advice about specific problems that can occur when you start an integration node, see the following sections.

- “Error messages AMQ7626 and BIP8048 are displayed when you try to start an integration node” on page 2899
- “Error messages BIP8048 and BIP8059 are displayed when you try to start an integration node” on page 2899

The integration node fails to start because there is not enough space in the Java TMPDIR directory or access permissions for the Java TMPDIR directory are inadequate

Procedure

- **Scenario:** The integration node fails to start and either an error message indicates that insufficient space is available or a BIP4512 exception indicates a java.lang.NoClassDefFoundError in the stack trace.
- **Explanation:** This error has two possible causes:
  - The integration node uses Java JAR files. When the integration node starts, the Java runtime environment extracts the JAR files into a temporary directory, Java TMPDIR. On Linux, and UNIX computers, the TMPDIR directory is typically `/tmp`; on Windows computers, it is `c:\temp`. If this directory is not large enough to hold the JAR files, the integration node does not start.
  - If the program is packaged as a PAR file, the user must have access to the system temporary directory and adequate space must be available in the temporary directory.
- **Solution:** Use one of the following methods to specify the location of this temporary JAR directory:
  - Use the environment variable `TMPDIR`.
  - Set the system property `java.io.tmpdir`.

  Allow at least 50 MB of space per integration server in this directory for IBM App Connect Enterprise components. You might need more space if you deploy large user-defined nodes or other JARs to the integration node. You should ensure that all the dependencies of the compute node class are deployed to the integration node.

Error message BIP2228 is issued when you try to start a second integration node on Linux or UNIX

Procedure

- **Scenario:** Error message BIP2228, which mentions `semctl` in the syslog, is displayed when you try to start a second integration node on Linux or UNIX.
- **Explanation:** This error typically indicates a permissions problem with a semaphore used by IBM App Connect Enterprise. A semaphore is created when the first integration node starts after a reboot (or after an initial installation), and only members of the primary group of the semaphore creator can access this semaphore. This problem is a consequence of the UNIX System V IPC primitives that are used by IBM App Connect Enterprise.

  The BIP2228 message is logged by any integration node that is started by a user who is not a member of the primary group of the semaphore creator. The integration node tries to access the semaphore, but fails with a permissions-related error. The integration node then terminates with the BIP2228 message.
- **Solution:** Avoid this problem by ensuring that all user IDs used to start IBM App Connect Enterprise have the same primary group. If this action is impractical, ensure that all user IDs are members of primary groups of all other user IDs. Contact your IBM Support Center for further assistance.

MQIsdp client connection is refused by the integration node

Procedure

- **Scenario:** When a new MQIsdp client tries to connect to the integration node, its connection is refused.
- **Explanation:** MQIsdp Client ID fields must be unique. If a client sends a CONN packet that contains the same Client ID as a currently connected client, the behavior is undefined.
- **Solution:** Ensure that Client IDs are unique.

When you start the integration node through the DataFlowEngine, it cycles continually

Procedure

- **Scenario:** When you start the integration node through the DataFlowEngine, it continually cycles, starts and stops, and errors BIP2801 and BIP2110 appear in the log:

```
Unable to load implementation file '/opt/IBM/DistHu b/v2/lib/libdhbNBIO.so',  rc=The file access
 permissions do not allow the specified action.
IBMibm Integration Bus internal program error.
```

- **Explanation:** The permissions on `/opt/IBM` have a value of 700, meaning that the integration node service user ID cannot read the disthub files.
- **Solution:** Set the permissions on `/opt/IBM` to 755, which is rwxr-xr-x.

The Java installation is at an incorrect level

Procedure

- **Scenario:** You issue the command to start an integration node , but the integration node does not start and the system log includes message BIP8892, for example:

```
Verification failed. The installed Java level 1.3.2 does not
   meet the required Java level 1.5.
```

- **Explanation:** The command performs a check on the level of the Java product installed on the computer to ensure that the Java product is at the required level.

  The check has failed, therefore the integration node is not started.
- **Solution:**
  - On distributed systems, you must use the JVM that is supplied with IBM App Connect Enterprise; no other Java product is supported. Check that you have run **mqsiprofile**, or (on Windows only) that you issued the **mqsistart** command from the correct command console.

    If you ran the profile command, check the settings of the environment variables in **mqsiprofile**; MQSI_JREPATH, PATH, and the appropriate library path environment variables for your operating system. Change these settings to point to the integrated JVM, and ensure that no other Java installation is in the path.

Error message BIP8875 is issued when you start an integration node

Procedure

- **Scenario:** You issue the **mqsistart** command to start an integration node, but the integration node does not start and the system log shows message BIP8875, for example:

```
The component verification for INODE has finished,
```

```
but one or more checks failed.
```

- **Explanation:** The command performs a series of checks to ensure that the integration node environment, IBM MQ queues, and Java are correct and accessible.

  One or more of the checks for the integration node identified in the message has failed, therefore the integration node is not started.

- **Solution:** Look in the system log, or in the Application log in the Event Viewer on Windows.

  Additional messages have been written before this message to indicate which checks have failed. All the checks are performed every time you issue **mqsistart**, therefore all errors are included in the log. Some messages are also returned when you run the command from the command line.

  Investigate the one or more errors that have been reported and check return codes and additional details. Look at the complete message content to check for typical causes of the error, and follow the advice given for the messages that you see in the log. View the complete message text in the Diagnostic Messages reference topics.

  For example, you might see one or more of the following messages:

  – BIP8875W: The component verification for '*component_name*' has finished, but one or more checks failed.
  – BIP8877W: The environment verification for component '*component_name*' has finished, but one or more checks failed.
  – BIP8883W: The IBM MQ verification for component '*component_name*' has finished, but one or more checks failed.
  – BIP8885E: Verification failed. Failed to connect to queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8887E: Verification failed for queue '*queue_name*' on queue manager '*queue_manager_name*' while issuing '*operation*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8888E: Verification failed. Failed to disconnect from queue manager '*queue_manager_name*'. MQRC: *return_code* MQCC: *completion_code*
  – BIP8892E: Verification failed. The installed Java level '*level_installed*' does not meet the required Java level '*level_supported*'.
  – BIP8893E: Verification failed for environment variable '*variable_name*'. Unable to access file '*file_name*' with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*.
  – BIP8895E: Verification failed. Environment variable '*variable_name*' is incorrect or missing.
  – BIP8896E: Verification failed. Unable to access the registry with user ID '*user_ID*'. Additional information for IBM support: *data1 data2*
  – BIP8897E: Verification failed. Environment variable '*variable_name*' does not match the component name '*component_name*'.
  – BIP8903E: Verification failed. The APF Authorization check failed for file '*file_name*'.
  – BIP8904E: Verification failed. Failed to start file '*file_name*' with return code '*return_code*' and errno '*error_number*'.

  If you cannot resolve the problems that are reported, and you receive a message such as BIP8893 that includes additional information, include these items in the information that you provide when you contact IBM Service.

Warning messages BIP8288-BIP8297 are shown in the syslog when you start an integration node

Procedure

- **Scenario:** Warning messages BIP8288-BIP8297 are shown in the syslog when you start an integration node on Linux and UNIX systems.

- **Explanation:** One or more problems were detected with the ODBC environment on Linux and UNIX systems.

  When you start an integration node by using the **mqsistart** command, the **mqsicvp** command is run automatically to check that the integration node environment is set up correctly. On Linux and UNIX systems, this command also verifies that the ODBC environment is configured correctly. Warning messages are written to the syslog in the following situations:

  - The file that is referenced by the ODBCINI environment variable does not exist, or the integration node does not have access to read or write to the file.
  - The ODBCSYSINI environment variable is not set.
  - The directory that is referenced by the ODBCSYSINI environment variable does not contain a file called `odbcinst.ini`, or the integration node does not have access to read or write to the file.
  - The IE02_PATH is not set.
- **Solution:** Examine the warning messages in the syslog. To view more information, run the **mqsicvp** command from the command line.

Error messages AMQ7626 and BIP8048 are displayed when you try to start an integration node

Procedure

- **Explanation:** You see these messages when using an integration node on a queue manager that is configured for global coordination with Oracle.
- **Solution:** Start the queue manager manually with the **-si** flag before starting the integration node.

Error messages BIP8048 and BIP8059 are displayed when you try to start an integration node

Procedure

- **Explanation:** You see these messages when starting an integration node that does not have an associated queue manager, or the queue manager is not started.
- **Solution:** If you do not have a queue manager associated with the integration node, create it. Start the queue manager by using the **strmqm** command, and then start the integration node.

Resolving problems when starting resources

Use the advice given here to help you resolve common problems that can arise when you start resources other than an integration node.

About this task

To learn how to resolve common problems when starting an integration node, see “Resolving problems when starting an integration node” on page 2895.

For advice about specific problems that can occur when you start resources, see the following sections.

- “Resources terminate during startup” on page 2900
- “Resources hang at startup on Windows” on page 2900
- “Error message BIP8048 is issued when you start a component” on page 2901
- “A "Not Found" error is issued when you click a link to a specific tutorial” on page 2901
- “Error message BIP0832 is issued on startup” on page 2902
- “Your integration servers restart repeatedly” on page 2902

- “After creating or changing a policy, you restart your integration server but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer” on page 2902
- “A device allocation error is issued” on page 2903
- “Windows fails to recognize IBM App Connect Enterprise digital signatures: "Unknown Publisher"” on page 2904
- “The create command fails, and error message BIP8022 is issued” on page 2905
- “Your integration server restarts repeatedly with a JVM Startup failure” on page 2905

Resources terminate during startup

Procedure

- Windows

  Scenario: The following error message is displayed when you start an integration node on Windows:

  ```
  ServiceName - DLL initialization failure Initialization of the
  dynamic link library c:\windows\system32\user32.dll failed.
  The process is terminating abnormally.
  ```

- **Explanation:** This error is issued by Windows when it fails to start a service because it has insufficient storage.
- **Solution:** This error is an operating system problem. Information about how to recover from this problem is available in the Microsoft Developer Network (MSDN). You can access MSDN on the Web at `http://msdn.microsoft.com`.

Resources hang at startup on Windows

Procedure

- Windows

  Scenario: You try to start the integration node on Windows, but nothing happens in the Event log to show that a connection has started.

- **Explanation:** This problem is typically caused by processes having only one thread. To see if this is the cause, check the Windows Task Manager. If either of the processes `bipconfigmgr.exe` or `dataflowengine.exe` has started, check the number of threads owned by the process. If the process has only one thread, you are likely to encounter this problem.

- **Solution:**
  a) Shut down the integration node using the **mqsistop** command and end the process from within the Task Manager.
  b) From the Windows **Start** button, click **Settings** > **Control Panel**
  c) Double-click **Administrative Tools**
  d) Double-click **Services** to open the **Services** window. From the list of available services, locate and right-click the integration node resource that you want to start (the service name begins with IBM App Connect Enterprise component). Click **Properties** from the menu.
  e) Make a note of the **This Account** setting. Contact the system administrator to obtain the password associated with **This Account**, because these settings are lost when you change values.
  f) Select **System Account** as the **Log On As** option, and select **Allow Service to Interact with Desktop.** These selections allow you to see any hidden dialog messages. Click **OK** to accept the changes.
  g) Restart the resource that is failing and report any subsequent error messages and dialog box messages to your IBM Service Representative.
  h) When your IBM Service Representative has resolved this problem for you, make sure that you restore the **This Account**, **Password**, and **Confirm Password** entries to the values that you used when you created the integration node.

Error message BIP8048 is issued when you start a component

Procedure

- **Scenario:** Error message BIP8048 is issued when you start a component.
- **Explanation:** This message indicates that IBM MQ is not responding as expected when it tries to start the queue manager. This problem might be because of the following reasons:
  - Linux UNIX The **strmqm** executable file is not present on Linux or UNIX systems.
  - Windows The **amqmdain** executable file is not present on Windows
- **Solution:** Check that your IBM MQ installation is fully functional:
  - On Windows, start the "IBM MQSeries" service.
  - On UNIX or Linux, issue the **strmqm** command to start the queue manager that is associated with this component.

  If the check fails, your IBM MQ installation is incomplete. This error occurs typically because you have previously installed WebSphere Application Server, which installs an embedded IBM MQ component that does not support IBM App Connect Enterprise.

  Uninstall WebSphere Application Server, then install the full IBM MQ product that is provided with IBM App Connect Enterprise.

A "Not Found" error is issued when you click a link to a specific tutorial

Procedure

- **Scenario:** You see a "Not Found" error when you click a link to a specific tutorial, indicating that a URL is invalid.
- **Explanation:** You can view tutorial applications only when you use the product documentation that is integrated with the IBM App Connect Enterprise Toolkit. If you are viewing a stand-alone or online product documentation, you cannot access these resources.

- **Solution:** If you want to access tutorials, ensure that you are viewing the product documentation from within the IBM App Connect Enterprise Toolkit.

Error message BIP0832 is issued on startup

Procedure

- **Scenario:** The following error message is displayed on startup:

  ```
  BIP0832E: A class java.io.FileNotFoundException exception occurred which
  reported the following message: [filepath] (The process cannot access the
  file because it is being used by another process). Resolve the reason of
  error and try again.
  ```

- **Explanation:** An invalid IBM MQ Java Client trace output file has been specified on the Enqueue preferences screen.
- **Solution:**
  a) Open the Enqueue preferences screen by clicking **Windows** > **Preferences**, then clicking **Enqueue** on the left.
  b) In the **To file** field, specify a valid output file (one that is not read-only or already in use).

Your integration servers restart repeatedly

Procedure

- **Scenario:** Your integration servers restart repeatedly. The system log might show an error, such as BIP2060.
- **Explanation:** The problem might be caused by:
  - The integration node environment variables incorrectly defined
  - Incorrect Loadable Implementation Library directory permissions
  - Incorrect database permissions
  - Invalid user-written LILs
- **Solution:** Check:
  - Environment variables as described in "Setting up a command environment" on page 64
  - Group memberships as described in Integration servers and integration nodes

After creating or changing a policy, you restart your integration server but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer

Procedure

- **Scenario:** After creating or changing a policy, you restart your integration server but your message flow does not start, and message BIP2275 is issued in the system log or Windows Event Viewer, indicating that an error occurred while loading the message flow from the persistent store.
- **Explanation:** When you change or create the policy, the connection properties are not fully validated at that point; the integration server does not attempt to use them to make a connection. For inbound adapters, the connection is made only when the associated message flow is deployed to the integration server and started. Therefore, the properties that you set on the policy might be invalid.

- **Solution:** Look at the messages following the BIP2275 message to determine if the message flow failed to start because of invalid connection properties.
  For example, in SAP you would see message BIP3414 with a reason such as:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE'
```

  followed by a BIP3450 message with an adapter error message such as:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE
```

  This error was detected by the adapter. The following message describes the diagnostic information that is provided by the adapter:

```
Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host
ERROR partner not reached (host invalidhost.test.co, service 3300)
TIME Fri Nov 28 15:27:32 2008
RELEASE 640
COMPONENT NI (network interface)
VERSION 37
RC -10
MODULE nixxi_r.cpp
LINE 8728
DETAIL NiPConnect2
SYSTEM CALL SiPeekPendConn
ERRNO 10061
ERRNO TE
```

  This message suggests that the **applicationServerHost** and **gatewayHost** properties are incorrect. When you have determined which properties are incorrect, use the **mqsichangeproperties** command to correct the properties.

A device allocation error is issued

Procedure

- **Scenario:** A device allocation error is issued.
- **Explanation:** A likely cause of this problem is that you do not have the correct permissions set on the component file system for the started task ID.

- **Solution:** Check the system log; if the problem is caused by having incorrect permissions set for the started task ID, you often see an RACF authorization failure message, as shown in the following example.

```
ICH408I USER(TASKID1 ) GROUP(TSOUSER ) NAME(FRED (FRED) 959
 /argo/MA11BRK/ENVFILE
 CL(DIRSRCH ) FID(01D7C7E2E3F0F8000F16000000000003)
 INSUFFICIENT AUTHORITY TO LOOKUP
 ACCESS INTENT(--X)  ACCESS ALLOWED(OTHER ---)
IEE132I START COMMAND DEVICE ALLOCATION ERROR
IEA989I SLIP TRAP ID=X33E MATCHED.  JOBNAME=*UNAVAIL, ASID=00A8.
D J,BPXAS
IEE115I 11.13.04 2001.212 ACTIVITY 601
```

  In this example, the started task ID does not have access to the file system component. The ICH408I message shows:

  – The file that the task is trying to access
  – The user ID that is trying to access the file
  – The permissions that the ID is expecting to have (INTENT in the message)
  – The permissions that the ID actually has (ALLOWED in the message)

  You can use this information to correct the permissions, then reissue, in this example, the request to start the integration node. This type of message is produced if the user who is issuing the command (which might be to start the integration node, or to submit JCL to start one of the utility jobs) does not have the correct file system permissions for the file system component. Use the ICH408I information to rectify the problem.

  Another possible reason for authorization failures is inconsistencies in the RACF definitions for a user ID in the MVS image and the OMVS segment. Also check with your system administrator that the RACF ID that is used on MVS has a corresponding OMVS image created.

Windows fails to recognize IBM App Connect Enterprise digital signatures: "Unknown Publisher"

Procedure

- **Scenario:** User A installs IBM App Connect Enterprise, and can run all programs (`mqsi*.exe`, `bip*.exe`, including the command console launcher). User B is created and given appropriate privileges. When User B runs an executable file such as the command console launcher, a window opens and reports that the executable file is from an unidentified publisher.
- **Explanation:** The operating system has not installed the appropriate digital certificates for User B.
- **Solution:** User B must manually install the certificates:

  1. In Windows Explorer, navigate to the `bin` directory for the IBM App Connect Enterprise installation; for example, on 32-bit systems, `C:\Program Files\IBM\MQSI\7.0\bin`
  2. Right-click any .exe file to open the **Properties** window.
  3. Click the **Digital Signatures** tab.
  4. Select the appropriate certificate from the list, then click **Details**. The **Digital Signature Details** window is displayed.
  5. Click **View Certificate**. The **Certificate** window is displayed.
  6. Click **Install Certificate** and complete the steps in the wizard. (Click **Next**, **Next**, **Finish**, then click **OK**.)
  7. Close the **Certificate** window. You return to the **Digital Signature Details**.
  8. Select the countersignature from the list, then click the **Details** button. A new **Digital Signature Details** window is displayed. You can repeat the preceding steps to install other certificates.

The create command fails, and error message BIP8022 is issued

Procedure

- **Scenario:** Error message BIP8022 is displayed when you use the **mqsicreatebroker** command on Windows, even if the supplied user name and password are correct.
- **Explanation:** The Microsoft component "Shared File and Printer Services" is required.
- **Solution:** Correct this error by installing the "Share File and Printer for Microsoft network" service on the Windows system.

Your integration server restarts repeatedly with a JVM Startup failure

Procedure

- **Scenario:** When you start the DataFlowEngine it continually starts and stops, displaying errors BIP2116E and BIP7409S in the log:

```
BIP2116E: IBM Integration Bus internal error: diagnostic information 'Fatal Error; exception
 thrown before initialisation completed', 'JVM Startup'
BIP7409S: The integration node was unable to create a JVM. The return code indicates that an
 unrecognized option was passed in to it.
```

- **Explanation:** When you start an integration server, it creates a Java virtual machine (JVM) for executing Java user-defined nodes, and its creation failed due to an incorrect JVM option.
- **Solution:** Correct the JVM option by completing the following steps:

  a) Stop the integration node.

  ```
  mqsistop integrationNodeName
  ```

  b) Check the jvmSystemProperty value of the failing integration server.

  ```
  mqsireportproperties brokerName -e egName -o ComIbmJVMManager -n
  jvmSystemProperty -f
  ```

  c) If the jvmSystemProperty has an invalid option, correct or reset its value.

  ```
  mqsichangeproperties brokerName -e egName -o ComIbmJVMManager -n
  jvmSystemProperty -v "" -f
  ```

  d) Start the integration node.

  ```
  mqsistart integrationNodeName
  ```

Resolving problems that occur when migrating or importing resources

Use the advice given here to help you to resolve common problems that can occur when you import or migrate resources.

About this task

Migration information is regularly updated on the IBM App Connect Enterprise support web page with the latest details available. Click **Troubleshoot**, then look for a document with a title like "Problems and solutions when migrating".

Procedure

- “Resolving problems when migrating or importing message flows” on page 2906
- “Resolving problems when migrating or importing message models” on page 2906
- “Resolving problems when migrating or importing other resources” on page 2908

Resolving problems when migrating or importing message flows

Use the advice given here to help you to resolve common problems that can occur when you import or migrate message flows.

Message flows that refer to a migrated user-defined node have connection errors

Procedure

- **Scenario:** After migration, all message flows that refer to a migrated user-defined node have errors indicating that connections cannot be made.
- **Explanation:** One possible cause is that the original user-defined node had space characters as part of one or more terminal names.

  The spaces are wrongly rendered as 'X20'.
- **Solution:**

  Edit the user-defined node `.msgnode` file, which is available in the same project as the flows that you have migrated. Correct any terminal names that are at fault. Ensure that the names are exactly as the message flow node implementation expects.

After migration, message flows cannot locate a user-defined node

Procedure

- **Scenario:** After migration, message flows cannot locate a user-defined node.
- **Explanation:** One possible cause is that the flows do not have the correct reference internally to a user-defined node.
- **Solution:** Select the **Locate subflow** menu for the node that cannot be located. Using the Browse dialog box, locate the user-defined node (which is in the same project as migrated flows).

  The message flow now links to the user-defined node correctly and the task list entry is removed when you save the flow.

Resolving problems when migrating or importing message models

Use the advice here to help you to resolve common problems that can occur when you import or migrate message models.

About this task

- "New instances of error message CTDV1534E" on page 2906
- "New instances of error messages CTDV1561E, CTDV1560E, CTDV1431E" on page 2907
- "New instances of error messages CTDV1150E, CTDV1118E, CTDV1432E, CTDV1446E, CTDV1466E, CTDV1467E" on page 2907
- "New instances of error message CTDV1625E" on page 2907
- "New instances of error message CTDV1458E" on page 2908
- "COBOL compiler errors when importing a copybook" on page 2908

New instances of error message CTDV1534E

Procedure

- **Scenario:** A DFDL schema that was created by a version of IBM App Connect Enterprise earlier than Version 10.0 is validated in the toolkit or during deployment. Extra instances of CTDV1534E are issued.

- **Explanation:** The IBM implementation of DFDL added a validation check to comply with the DFDL 1.0 specification. The DFDL property 'length' of an element must not exceed the capacity of the element's simple type. This check was being made for DFDL 'lengthUnits' = 'bits', it is now also being made for 'lengthUnits' = 'bytes'. The addition of this check can cause extra instances of CTDV1534E.
- **Solution:** Change the element's simple type so that it can cope with the length. For example, a two's complement binary number has DFDL 'length' = '8' and type is xs:int. Change the type to xs:long to clear the error.

New instances of error messages CTDV1561E, CTDV1560E, CTDV1431E

Procedure

- **Scenario:** A DFDL schema that was created by a version of IBM App Connect Enterprise earlier than Version 10.0 is validated in the toolkit or during deployment. Extra instances of CTDV1561E, CTDV1560E, CTDV1431E are issued.
- **Explanation:** The IBM implementation of DFDL added validation checks to comply with the DFDL 1.0 specification. Checks need to be made on elements and groups when DFDL property 'initiatedContent' = 'yes' on the parent sequence or choice. Some of these checks were missing, and have now been added. The addition of these checks can cause extra instances of CTDV1561E, CTDV1560E, or CTDV1431E.
- **Solution:** Set 'initiatedContent' = 'no' on the parent sequence or choice. If a new error CTDV1559E then appears, contact your IBM Support Center for further advice.

New instances of error messages CTDV1150E, CTDV1118E, CTDV1432E, CTDV1446E, CTDV1466E, CTDV1467E

Procedure

- **Scenario:** A DFDL schema that was created by a version of IBM App Connect Enterprise earlier than Version 10.0 is validated in the toolkit or during deployment. Extra instances of CTDV1150E, CTDV1118E, CTDV1432E, CTDV1446E, CTDV1466E, CTDV1467E are issued.
- **Explanation:** The IBM implementation of DFDL added validation checks to comply with the DFDL 1.0 specification. Cross-checks need to be made between DFDL properties of elements and groups and DFDL properties on the parent sequence or choice. Some of these checks were missing when the parent sequence or choice was the content of a global group and DFDL properties were placed on group references to the group. These checks have now been added. The addition of these checks can cause extra instances of CTDV1150E, CTDV1118E, CTDV1432E, CTDV1446E, CTDV1466E, or CTDV1467E.
- **Solution:** Correct the schema in accordance with the indicated error.

New instances of error message CTDV1625E

Procedure

- **Scenario:** A DFDL schema,that was created by a version of IBM App Connect Enterprise earlier than Version 10.0 is validated in the toolkit or during deployment. Extra instances of CTDV1625E are issued.
- **Explanation:** The IBM implementation of DFDL added validation checks to comply with the DFDL 1.0 specification. When the DFDL property 'occursCountKind' of an element is 'parsed' and the parent sequence has a separator, the DFDL property 'separatorSuppressionPolicy' of the parent sequence must be 'anyEmpty'. This new check is the result of a specification erratum.
- **Solution:** Correct the sequence so DFDL 'separatorSuppressionPolicy' is 'anyEmpty', or change the element so DFDL 'occursCountKind' is 'implicit'.

New instances of error message CTDV1458E

Procedure

- **Scenario:** A DFDL schema,that was created by a version of IBM App Connect Enterprise earlier than Version 10.0 is validated in the toolkit or during deployment. Extra instances of CTDV1458E are issued.
- **Explanation:** The IBM implementation of DFDL has added validation checks to comply with the DFDL 1.0 specification. The DFDL property '`fillByte`' must resolve to a single byte value. This check was not being made correctly when '`fillByte`' contained DFDL entities such as `%r00;`. The corrected check can cause extra instances of CTDV1458E.
- **Solution:** Correct the '`fillByte`' to use a single DFDL entity that resolves to a single byte value.

COBOL compiler errors when importing a copybook

Procedure

- **Scenario:** The report file that is generated by the import contains COBOL compiler errors.

  For example, you try to import the following copybook:

  01 AIRLINE-REQUEST.
  ....05 CUSTOMER.
  ....10 NAME.....PIC X(45).
  ....05 ADDRESS.
  ....10 STREET.....PIC X(30).
  ....10 CITY.....PIC X(25).
  ....10 STATE.....PIC X(20).
  ....10 ZIP-CODE.....PIC X(5).
  ....05 FLIGHT-NO.....PIC X(6).
  ....05 TRAN-DATE.....PIC X(10).
  ....05 COST.....PIC X(7).
  ....05 CC-NO.....PIC X(20).
  ....05 RESPONSE.
  ....10 STATUS.....PIC X(100).
  ....10 DETAILS.....PIC X(100).

  The report file contains errors:

```
Line No : 4 IGYDS1089-S "ADDRESS" was invalid. Scanning was resumed at the next area "A"
 item, level-number, or the start of the next clause.
Line No : 14 IGYDS1089-S "STATUS" was invalid. Scanning was resumed at the next area "A"
 item, level-number, or the start of the next clause.
```

- **Explanation:** The errors are caused by the copybook containing field names that are COBOL reserved keywords.
- **Solution:** Change the name of the fields in question, so that they are not COBOL reserved keywords, and retry the import.

Resolving problems when migrating or importing other resources

Use the advice given here to help you to resolve common problems that can arise when you import or migrate resources other than message flows.

About this task

- "You see an error message when you recompile a BAR file from a previous version" on page 2909
-

- "The FileImport menu provides only the option to import a compressed file inside an existing project" on page 2909

You see an error message when you recompile a BAR file from a previous version

Procedure

- **Scenario:** You have imported a BAR file with your resources from a previous version, and you then choose to refactor those resources to applications and libraries. You try to recompile a BAR file after the resources have been migrated to applications and libraries, but you see an error message similar to the following example:

```
TotalPurchaseOrderFlow.msgflow belongs in an application or library and should be deployed
 within that container and not independently.
Create a new BAR file and select the application or library in the Prepare tab of the BAR
 editor, then select Build and Save.
To deploy the resource separately from the application or library, it must be moved into a
 Message Broker project.
```

- **Explanation:** If you have reorganized your imported resources into applications and libraries, you cannot rebuild the original BAR file. If a message flow from your original BAR file has been moved into an application in IBM App Connect Enterprise, you must deploy the flow with the new container, or move it to an integration project, from which you can deploy it separately.
- **Solution:** Create a new BAR file and add the application or library that contains the resources that you want to deploy. To deploy a resource like a message flow on its own, move the flow to an integration project, then deploy the flow separately.

The File > Import menu provides only the option to import a compressed file inside an existing project

Procedure

- **Scenario:** You have a compressed file that contains message set projects and message flow projects. When you click **File** > **Import**, you have only the option to import the compressed file inside an existing project, but you want to re-create the message set projects and message flow projects.
- **Solution:** When you export and import files, do not export or import the root directory, which is created for you because of the project file. When you export your message flow and message set projects:
  a) Click **Create only selected directories**.
  b) Clear the project root folder.
  c) Select the files and subdirectories as required.
  The project root folder is selected, but is displayed as gray.

  Then, when you import the compressed file:
  a) Clear the root (/) folder.
  b) Select the files and subfolders as required.
  The project root folder is selected, but is displayed as gray.

Resolving problems when stopping resources

Use the advice given here to help you to resolve problems when you stop resources.

About this task

Procedure

- "You cannot stop the integration node" on page 2910
- "You cannot stop the integration node queue manager" on page 2910

- "The integration server ends abnormally" on page 2910

You cannot stop the integration node

Procedure

- **Scenario:** You run the **mqsistop** command to stop the integration node, but the system freezes, and does not stop any of the integration servers.
- **Explanation:** One possible cause is that a message flow is being debugged and it is currently stopped at a breakpoint. IBM App Connect Enterprise regards this as a message in flight situation, and refuses to stop the integration node through the normal command.
- **Solution:** Click **Stop debugging** in the Integration Development perspective of the IBM App Connect Enterprise Toolkit. After that operation has completed, the integration node stops.

  If you cannot stop the debug session, end all integration server processes that are associated with that integration node to allow the integration node to stop. Your messages are backed out. Click **Stop debugging** after the integration node restarts.

You cannot stop the integration node queue manager

Procedure

- **Scenario:** You are trying to use the IBM MQ **endmqm** command to stop an integration node queue manager on a distributed system, but it does not stop.
- **Explanation:** In certain circumstances, attempting to stop an integration node queue manager does not cause the queue manager to stop. This situation can occur if you have configured any message flows with multiple threads (you have set the message flow property Additional Instances to a number greater than zero).
- **Solution:** If you want to stop the integration node's queue manager, stop the integration node first by running the **mqsistop** command, and then stop the queue manager.

The integration server ends abnormally

About this task

Procedure

- **Scenario:** Your integration server processes end abnormally.
- **Explanation:** When integration server processes end abnormally, they are restarted automatically by the `bipbroker` process. If an integration server process fails, it is restarted three times during each five-minute interval. The first five-minute interval begins when the integration server is first started. RetryInterval defaults to 5

  Remove the integration server from the integration node configuration, deploy the integration node configuration, then later add the integration server, and redeploy the integration node configuration. The row is re-created and RetryInterval is set to its default value of 5.
- **Solution:** To change the default value:

  a) Stop the integration node.

  b) Change the value of the RetryInterval in the database table.

  c) Restart the integration node.

Resolving problems when deleting resources

Use the advice given here to help you to resolve problems when you delete resources.

About this task

The Windows service that is associated with an integration node is not deleted when the integration node is deleted

Procedure

- **Scenario:** When you delete an integration node on Windows, the Windows service that is associated with the integration node might remain in a "Pending deletion" state.
- **Explanation:** The "Pending deletion" state can occur if a Windows system tool has a handle on the service.
- **Solution:** Close the system tool that has a handle on the Windows service, and the service should be deleted.

You cannot delete a project from your workspace

Procedure

- **Scenario:** You cannot delete a project from your workspace. You get error messages indicating that the containing directory cannot be deleted, or the project file is missing.
- **Explanation:** If you attempt to delete a project, and the directory that contains the project is in use, or you have any files that are contained within the project that have been opened by programs other than the IBM App Connect Enterprise Toolkit, some of the resources in the project are not deleted, but others, including the project file, might be deleted.
- **Solution:** Before you delete a project, make sure that other applications do not have the files open, and that you do not have an open command prompt located in the directory. To recover from this problem, manually delete any remaining files and directories from your workspace directory, then click **Delete** from the project in the IBM App Connect Enterprise Toolkit.

Resolving problems when developing message flows

Use the advice given here to help you to resolve common problems that can arise when developing message flows.

About this task

- “Resolving appearance problems when developing message flows” on page 2912
- “Resolving problems when you use callable flow nodes” on page 2912
- “Resolving problems when you use CORBA nodes” on page 2915
- “Resolving problems when you use Email nodes” on page 2916
- “Resolving ESQL problems when developing message flows” on page 2918
- “Problems when developing message flows with file nodes” on page 2920
- “Resolving problems when you use HTTP and SOAP nodes” on page 2924
- “Resolving implementation problems when developing message flows” on page 2927
- “Resolving problems when you use IMS nodes” on page 2936
- “Resolving problems when you use Salesforce nodes ” on page 2939
- “Resolving problems when using Kafka nodes ” on page 2941
- “Resolving mapping and message reference problems when developing message flows” on page 2943

- "Resolving trace problems when developing message flows" on page 2944
- "Resolving problems when developing message flows with WebSphere Adapters nodes" on page 2945
- "Resolving other problems when developing message flows" on page 2951

Resolving appearance problems when developing message flows

This topic contains advice for dealing with some common appearance problems that can arise when developing message flows:

The task list does not update when you make corrections to your files

Procedure

- **Scenario:** The task list does not update any modifications that you make to an ESQL or mapping file. You have made corrections to the files, and while there are no error flags in the file or file icon, the error remains as a task list item.
- **Solution:** To work around this problem, set the following environment variable:

```
JITC_COMPILEOPT=SKIP{org/eclipse/ui/views/tasklist/TaskListContentProvider}
{resourceChanged}
```

You rename a flow that contains errors, but the task list entries remain

Procedure

- **Scenario:** When you rename a message flow in the IBM App Connect Enterprise Toolkit for which there are error icons (red crosses) displayed on nodes and connections, those error icons are removed when changes are made. However, the task list entries remain.
- **Solution:** Refresh the Message Flow editor by closing and reopening it.

Terminals on a subflow get out of sync as changes are made

Procedure

- **Scenario:** You have a message flow that contains subflow nodes. The name or number of terminals on the subflow gets out of sync when changes are made on the subflow itself. The same problem can happen with promoted properties.
- **Solution:** Refresh the Message Flow editor by closing and reopening it. Close the Message Flow editor that contains subflows while the subflows are being changed.

Resolving problems when you use callable flow nodes

Diagnose problems with callable flows by validating messages and running commands.

Before you begin

Read the concept information in "Callable message flows" on page 491.

About this task

To check the current operational status of IBM App Connect on IBM Cloud, see the status page: IBM App Connect status.

If problems occur when you develop callable flows, complete the following checks.

For all callable flows:

- "Validate messages before they reach the CallableFlowInvoke node" on page 2913

- “Ensure that application and endpoint name pairs are unique on a single integration server” on page 2913

For callable flows that are split between IBM App Connect Enterprise and IBM App Connect on IBM Cloud:

- “Check that the Switch server is running” on page 2913
- “Check that your callable flows are registered” on page 2914
- “Determine the status of the switch server for callable flows” on page 2914
-

Validate messages before they reach the CallableFlowInvoke node

About this task

The CallableFlowInvoke node parses the incoming message in full so that it is in a suitable format to send to the CallableInput node. Therefore, you should validate the message before it reaches the CallableFlowInvoke node. If the call fails due to a parsing failure, the message is rolled back with an exception.

Binary Large Object (BLOB) messages do not need to be parsed. Therefore, if you do not want to parse the message data before the CallableFlowInvoke node, you can send data to the callable flow by using the BLOB parser.

Ensure that application and endpoint name pairs are unique on a single integration server

About this task

Application and endpoint name pairs must be unique on a single integration server. You can have multiple callable flows that share the same application and endpoint names, but they must be in different integration servers. In this case, the Switch server acts as a load balancer.

Similarly, if you are splitting processing between IBM App Connect Enterprise and IBM App Connect on IBM Cloud, any callable flows with the same application and endpoint names must be in different integrations in the cloud.

Check that the Switch server is running

About this task

The Switch server is a special integration server that runs in a special integration node called IIBSWITCH_NODE. This server routes data between your callable flows. You cannot deploy anything to this server. If your callable flows are deployed to different integration servers, you run the **iibcreateswitchcfg** command to create configuration files for the Switch server and for the agents that enable your flows to use the Switch server securely. You run the **iibswitch** command to create and start the Switch server by using the generated configuration file (`switch.json`).

After you run the **iibswitch** command, check that the Switch server is created and running by using the **mqsilist** command:

```
mqsilist IIBSWITCH_NODE
```

If the Switch server is running, you see the following response:

```
BIP1286I: Integration server 'IIBSWITCH_SERVER' on integration node 'IIBSWITCH_NODE' is running.
```

If the Switch server is stopped, start it by running the following command:

```
iibswitch /start switch
```

If the Switch server is not created, run the following commands:

```
iibcreateswitchcfg /output filepath
iibswitch /create switch /config filepath\switch.json
```

Determine the status of the switch server for callable flows

About this task

You can view the status of the switch server by running the following command (do not replace IIBSWITCH_NODE or IIBSWITCH_SERVER with the name of an integration node or integration server):

```
mqsireportproperties IIBSWITCH_NODE -e IIBSWITCH_SERVER -o ComIbmIIBSwitchManager/Switch -r
```

The output of this command also includes details of the callable flows that have been registered in this switch server, and the details of the callable flow invoke nodes that have been registered in this switch server.

Check that your callable flows are registered

About this task

The **mqsireportproperties** command lists the calling and callable flows that are registered for a specified integration server. Run the following **mqsireportproperties** command for each integration server:

```
mqsireportproperties integrationNodeName -e integrationServerName -o CallableFlowManager -r
```

The following example shows a typical response:

```
callable-flow-manager
  name='CallableFlowManager'
  identifier='CallableFlowManager'
  type='CallableFlowManager'
  delayReplyUntilFlowCommit='false'
  disableRemoteInputRegistration='false'
  redeployMessageTimeout='5'
  remoteRetryTimeout='60'
  retryLocalDuringDeploy='true'
  retryRemoteUntilTimeout='true'
  detailed
    registrations
      callableFlow
        applicationName='app2'
        endpointName='callableEndpoint1'
        inputNodeRegistered='0x0000000006AF8480'
        invokeNodeCount='1'
        queueDepth='0'
      callableFlow
        applicationName='app4'
        endpointName='callableEndpoint2'
        inputNodeRegistered='0x0000000000000000'
        invokeNodeCount='1'
        queueDepth='0'
```

A message flow that calls a callable flow contains a CallableFlowInvoke node. Therefore, if a calling flow is deployed to the server, the invoke node count is 1. The **mqsireportproperties** command also specifies the application name and endpoint name that the calling flow is using to call the callable node.

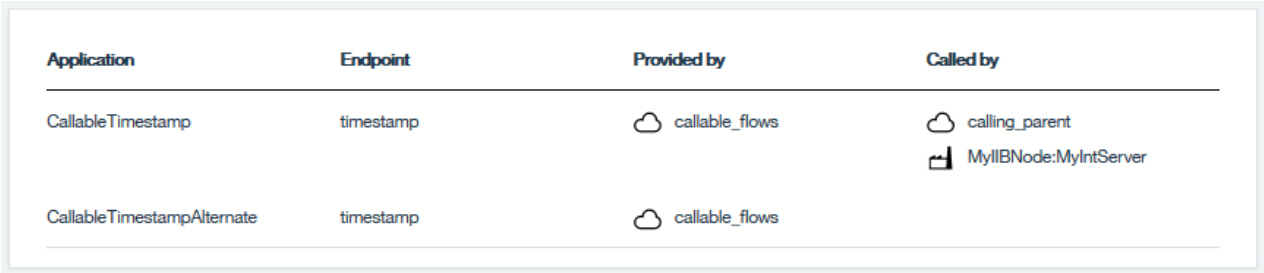
A callable flow contains a CallableInput node. If a callable flow is deployed to the server, the inputNodeRegistered value is greater than zero. The **mqsireportproperties** command also specifies the name of the application that contains the callable flow, and the endpoint name that is set on the CallableInput node of the callable flow.

If both the calling and the callable flow are deployed to the same integration server, the invokeNodeCount value is 1, and the inputNodeRegistered has a value that is greater than zero. In the example, the inputNodeRegistered value indicates that a callable flow with an endpoint name

of *callableEndpoint1* is deployed to this server in an application called *app2*. The `invokeNodeCount` value of 1 indicates that the flow that calls this callable flow is also deployed to this server. However, the `inputNodeRegistered` value is zero for the callable flow in application *app4* with an endpoint name of *callableEndpoint2*, but the invoke node count is 1. Therefore, the calling flow is deployed to this server, but the callable flow is not.

Using the Callable Flows view in IBM App Connect on IBM Cloud

If one of your callable flows is in IBM App Connect on IBM Cloud, you can check that your flows are registered with the Switch server in the Callable flows view. This view lists the callable flows that are running in IBM App Connect on IBM Cloud.



This example shows that a callable flow called *CallableTimestamp* is deployed to IBM App Connect on IBM Cloud in an integration server called *callable_flows*. This callable flow is called by a flow on another integration server in IBM App Connect on IBM Cloud, called *calling_parent*. The same flow is also called by an on-premises flow that is deployed to the integration server *MyIntServer* on the integration node *MyIIBNode*. You can also see the endpoint name that is used to link the calling and called flows.

If your on-premises flows are not listed in the Callable flows view, restart the IBM App Connect Enterprise integration server, then refresh the IBM App Connect on IBM Cloud web page in your browser.

Resolving problems when you use CORBA nodes

Advice for dealing with common problems that can arise when you develop message flows that contain CORBA nodes.

Before you begin

IBM App Connect Enterprise does not currently support all CORBA operations and types. Ensure that you are passing in a valid IDL file that contains supported operations and types. For a full list of what is supported, see "CORBA support" on page 1155. To ensure that the IDL file is valid, run it through an IDL parser.

About this task

Error message BIP4891 is issued when you include a CORBARequest node in a message flow

Procedure

- **Scenario:** You have created a message flow that contains a CORBARequest node, but error message BIP4891 is issued, indicating that the node did not receive a valid body.
- **Explanation:** This error message indicates that the CORBARequest node is trying to call an operation but cannot find the required input parameters in the incoming tree. IBM App Connect Enterprise uses

the DataObject parser to read and write message from CORBA applications. If you use an input node to pass XML into the CORBARequest node, ensure that it uses the DataObject domain.

- **Solution:** Ensure that the incoming message has the correct structure. If you are using an input node to pass XML into the CORBARequest node, set the `Message domain` property on the **Input Message Parsing** tab of the input node to `DataObject`.

A CORBA IDL file drop error is issued when you are using an IDL file that contains includes

Procedure

- **Scenario:** You have dragged a CORBA IDL file onto the canvas but a CORBA IDL file drop error is issued.
- **Explanation:** If you have imported an IDL file that contains includes, you must drag the top-level IDL file onto the canvas so that the CORBARequest node has all the relevant information. Similarly, when setting properties on the CORBARequest node, if you have imported an IDL file that contains includes, you must select the top-level IDL file in the `IDL file` property.
- **Solution:** Drag the top-level IDL file onto the canvas, or set the `IDL file` property on the CORBARequest node to the top-level IDL file.

Error message BIP4910 is issued during deployment when you are using an IDL file that contains includes

Procedure

- **Scenario:** You are deploying a message flow that contains a CORBARequest node, but error message BIP4910 is issued.
- **Explanation:** This error is issued when you have imported an IDL file that contains includes, but not all the included IDL files have been added to the BAR file. For example, you might have dragged only the message flow onto the integration server. If you have imported an IDL file that contains includes, you must ensure that all the included IDL files are added to the BAR file so that all the relevant information is available to the message flow.
- **Solution:** When you deploy a message flow that contains a CORBARequest node and an IDL file that contains includes, ensure that all included IDL files are added to the BAR file.

  If you are dragging a message flow that uses a multifile IDL file onto an integration server in the Integration Development perspective, included IDL files are not deployed. To deploy message flows that use multifile IDL files, you must create a BAR file.

Resolving problems when you use Email nodes

Advice for dealing with common problems that can arise when you develop message flows that contain Email nodes.

About this task

- "A negative email size displays in the local environment" on page 2917
- "A parsing error displays when you reparse an email attachment as XML" on page 2917
- "Removing unwanted null characters from an email" on page 2917

A negative email size displays in the local environment

Procedure

- **Scenario:** The EmailInput node receives an email from an email server that supports Post Office Protocol 3 (POP3) but the size of the email, including any attachments, might display a negative value in the `Root.EmailInputHeader.Size` Multipurpose Internet Mail Extensions (MIME) logical tree.
- **Explanation:** The email server provider that supports POP3 uses the **TOP** command to fetch the headers for the email message and the **LIST** command to determine the size of the entire message. The server then subtracts the two values to determine the size of the message body. If the server reports the size of the entire message incorrectly, you might see a negative number in the local environment Size field.
- **Solution:** You can use a Compute node to calculate the size of the email message and the size of any attachments. The following example ESQL can be used to calculate the size of the email content and attachments for a multipart MIME document. In this example, the result is stored in the LocalEnvironment:

```
DECLARE CURSOR REFERENCE TO InputRoot.MIME.Parts;
      DECLARE I INTEGER 0;

      FOR SOURCE AS CURSOR.Part[] DO
         SET I = I + LENGTH( SOURCE.Data.BLOB.BLOB);
      END FOR;

      SET OutputLocalEnvironment.Variables.EmailSize = I;
```

  For more information about messages that belong to the MIME domain, see "Manipulating messages in the MIME domain" on page 1742.

A parsing error displays when you reparse an email attachment as XML

Procedure

- **Scenario:** Your message flow retrieves emails from an email server by using an EmailInput node. The email contains an XML document attachment that you want to reparse. However, when you try to reparse the attachment you receive parsing errors from IBM App Connect Enterprise reporting that you have an invalid XML character.
- **Explanation:** Some email servers might insert carriage return (CR) and line feed (LF) characters at the end of an email. Typically you would want to keep these characters, but in this scenario you must remove them so that you can reparse your XML data.
- **Solution:** Use the following ESQL in a Compute node to remove the CR and LF characters:

```
DECLARE NEWEMAIL BLOB TRIM( TRAILING X'0d0a' FROM InputRoot.
MIME.Data.BLOB.BLOB );
```

Removing unwanted null characters from an email

Procedure

- **Scenario:** Your email attachment contains unwanted null characters that you would like to remove.
- **Explanation:** Your email attachment might contain null characters that you would like to remove; for example, you intend to reparse the data in the attachment.
- **Solution:** Use the following ESQL in a Compute node to remove the null characters:

```
DECLARE NEWEMAIL BLOB TRIM( TRAILING X'00' FROM InputRoot.MIME.
Data.BLOB.BLOB)
```

Resolving ESQL problems when developing message flows

This topic contains advice for dealing with some common ESQL problems that can arise when developing message flows:

A `Routine not defined` error message is issued in ESQL when you move a routine

Procedure

- **Scenario:** A `Routine not defined` error message is displayed in ESQL when you move a routine from one schema to another.
- **Explanation:** If a routine that was referenced by code in one schema is moved to another schema, where it is still visible, a false error is generated stating that the routine cannot be resolved.
- **Solution:** Clean the project by clicking **Project** > **Clean**.

The product fails to respond when you paste ESQL statements from Adobe Reader

Procedure

- **Scenario:** When you copy and paste certain ESQL statements from Adobe Reader into the ESQL editor, IBM App Connect Enterprise stops responding.
- **Explanation:** This problem occurs when you paste text directly from Adobe Reader into either the ESQL editor or the Java editor.
- **Solution:** To work around this problem, either enter the text manually, or copy and paste it to a text editor (such as Notepad), then perform another copy and paste action from there.

You do not know how message flows handle the code page of ESQL files

Procedure

- **Scenario:** You do not know how message flows handle the code page of ESQL files.
- **Solution:** The code page of an ESQL file is the code page of the IBM App Connect Enterprise Toolkit on which the file is created. You must deploy a message flow using an ESQL file on an IBM App Connect Enterprise Toolkit with the same code page setting as the ESQL file. When multiple ESQL files are involved in a single compiled message flow (`.cmf`) file, all these ESQL files must be in the same code page.

  See Editor preferences and localized settings for more information.

You do not know the naming restrictions for ESQL procedures and functions

Procedure

- **Scenario:** You do not know the restrictions for choosing names for ESQL modules or schema scope ESQL and mapping procedures and functions.
- **Solution:** Module and schema scope procedures and functions cannot have names starting with `IBM_WBIMB_` because `IBM_` is reserved for IBM use, and `IBM_WBIMB_` is reserved for IBM App Connect Enterprise.

Error message BIP5431 is issued and the integration node fails

Procedure

- **Scenario:** Error message BIP5431 is displayed and the integration node fails.
- **Explanation:** When setting output message properties, you have specified an incorrect physical format name for the message format.

- **Solution:** The name that you specify for the physical layer must match the name that you have defined for it. The default physical layer names are `Binary1`, `XML1` and `Text1`.

You are unable to call Java from ESQL

Procedure

- **Scenario:** Your Java class files are not being found.
- **Explanation:** When creating the class files, you have not placed them in the correct location within the system CLASSPATH.
- **Solution:** See the CREATE PROCEDURE statement for further information.

Error message BIP3203 is issued: Format expression is not a valid FORMAT expression for converting expression to type

Procedure

- **Scenario:** Your format expression contains an unrecognized character for the conversion.
- **Explanation:** Your format expression for a numeric conversion was used to convert to or from a *DATE*, *TIME*, *TIMESTAMP*, *GMTTIME* or *GMTTIMESTAMP* variable. Another possible explanation is that your format expression for a DateTime conversion was used to convert to or from an *INTEGER*, *DECIMAL* or *FLOAT* variable.
- **Solution:** Replace the format expression with one from the applicable types. For more information about valid data types and expressions, see the ESQL reference topic.

Error message BIP3204 is issued: Input expression does not match FORMAT expression. Parsing failed to match

Procedure

- **Scenario:** You have used an input string that does not match the format expression.
- **Explanation:** Your format expression contains data that does not match the current element of the format expression.
- **Solution:** Either rewrite the format expression to match the input data, or modify the input data to match the format expression. For more information about valid data types and expressions, see the ESQL reference topic.

The CAST function does not provide the expected DST offset for non-GMT time zones

Procedure

- **Scenario:** You are using the CAST function to convert a string to a TIME variable, in an integration node that is running in a time zone other than GMT. The daylight saving time (DST) offset is not correctly calculated.
- **Explanation:** If no time zone is associated with the time string passed to CAST, it is converted to GMT time. If no date is supplied, the current system date is assumed.
- **Solution:** Specify the correct time zone and date. See Formatting and parsing dateTimes as strings for more information.

Error message BIP3205 is issued: The use of a FORMAT expression is not allowed when converting

Procedure

- **Scenario:** You have used a format expression when it is not applicable, for example when converting from decimal to integer.
- **Explanation:** The use of format expressions is limited to casting between datetime and string values or numeric and string values. Your format expression cannot be applied in this case.
- **Solution:** Either remove the FORMAT clause, or change the parameters. For more information about valid data types and expressions, see the ESQL reference topic.

Problems when developing message flows with file nodes

Use the advice given here to help you to resolve some common problems that can arise when you develop message flows that contain file nodes.

About this task

- "A file node flow stops processing files and error message BIP3331 or BIP3332 is issued" on page 2920
- "During processing of a large file, error message BIP2106 is issued or the integration node stops because of insufficient memory" on page 2921
- "Missing or duplicate messages after recovery from failure in a flow attached to a FileInput node" on page 2922
- "No file is created in the output directory after FileOutput node processing" on page 2922
- "Output file name overrides have not been applied" on page 2923

A file node flow stops processing files and error message BIP3331 or BIP3332 is issued

Procedure

- **Scenario:** Files in the specified input directory are not being processed. Error message BIP3331 or BIP3332 is issued.
- **Explanation:** The error messages explain that the FileInput node encountered an exception and could not continue file processing.

  This problem can be caused when the FileInput node cannot move files from its input directory to the archive or backout directory because of file system permissions or another file in the target directory preventing the file to be transferred. In this situation, the node is unable to process input files without losing data so processing stops. Two messages are issued; the first is either BIP3331 or BIP3332 which specifies a second message which describes the cause of the problem in more detail.

- **Solution:** If the first error message issued is BIP3331, stop the flow and resolve the problem. The FileInput node is unable to complete successful processing of the file.

  1. Stop the flow.
  2. Find the error message referenced in the BIP3331. This second error message identifies the problem and the files and directories causing it.
  3. Ensure the integration node has the required access to these files and directories.
  4. You might need to move, delete or rename files in the archive or transit directories.
  5. Check whether the input file causing the problem has been successfully processed (except for being moved to the archive or backout directory). If it has been successfully processed, remove it from the input directory.
  6. Restart the flow.

  If the first error message issued is BIP3332, you do not need to stop the flow because the FileInput node has detected the problem before starting file processing. Find the error message referenced in the BIP3332 message. This second error message identifies the problem and the files and directories causing it.

During processing of a large file, error message BIP2106 is issued or the integration node stops because of insufficient memory

Procedure

- **Scenario:** Large input files cause the integration node to issue messages, or stop, because insufficient memory is available.
- **Explanation:** The FileInput node can process very large files. Subsequent processing in the flow attached to its Out terminal might require more memory than is available to the integration node.
- **Solution:** In most cases, the FileInput node imposes a limit of 100 MB on the records propagated to the attached flow. If your application needs to access large amounts of data, you might need to increase its available memory and reduce the number of available instances. See “Resolving problems with performance” on page 3010 for more information.

  If your application needs to process messages larger than 100 MB, you can override the FileInput node record size limit by taking the following actions:

  – Before starting the integration node, set the environment variable MQSI_FILENODES_MAXIMUM_RECORD_LENGTH to the required limit as an integer number of bytes, for example:

  ```
  SET MQSI_FILENODES_MAXIMUM_RECORD_LENGTH=268435456
  ```

  – When the integration node first initializes a FileInput node, it will use the environment variable value instead of the default value of 100 MB. Subsequent changes to the environment variable value will not affect the integration node limit until the integration node is restarted.

  – If the `Record detection` property is set to `Whole File`, the limit applies to the file size. If the `Record detection` property is set to `Fixed Length` or `Delimited`, the limit applies to the record size. The FileOutput node is not affected by changes to this limit.

  – **Note:** Increasing the FileInput node record size limit might require additional integration node resources, particularly memory. You must thoroughly test and evaluate your integration node's

performance when processing these files. The number of factors that are involved in handling large messages make it impossible to provide specific integration node memory requirements.

You can also reduce the memory required to process the file's contents in the following ways:

– If you are processing a whole file as a single BLOB, split it into smaller messages by specifying on the **Records and Elements** tab of the FileInput node's properties:

  - A value of `Fixed Length` in the `Record detection` property
  - A large value in the `Length` property, for example 1000000.

– If you are writing the file's contents to a single output file, specify `Record is Unmodified Data` in the FileOutput node's `Record definition` property; this reassembles the records in an output file of the same size as the input file. Wire the FileInput node's End of Data terminal to the FileOutput node's Finish File terminal. Configure the flow to have no additional instances to ensure that the output records arrive in sequence.

– If you are processing large records using the techniques shown in the Large Messaging sample, ensure that you do not cause the integration server to access the whole record. Avoid specifying a value of $Body in the `Pattern` property of a Trace node.

– If you have specified a value of `Parsed Record Sequence` in the FileInput node's `Record definition` property, the integration node does not limit the size of the record. If subsequent nodes in the message flow try to access an entire large record, the integration node might not have sufficient memory to allow this and stop.

Missing or duplicate messages after recovery from failure in a flow attached to a FileInput node

Procedure

- **Scenario:** After the failure of a message flow containing a FileInput node processing the input file as multiple records, a subsequent restart of the flow results in duplicate messages being processed. If the flow is not restarted, some input records are not processed.
- **Explanation:** If a record produces a message which causes the flow to fail and retry processing does not solve the problem, the node stops processing the file and moves it to the backout directory. Records subsequent to the failing message are not processed. The FileInput node is not transactional; it cannot roll back the file input records. Transactional resources in the attached flow can roll back the effects of the failing input record but not preceding records. Records before the failing record will have been processed but records subsequent to the failing record will not have been processed. If you restart the flow by moving the input file from the backout directory to the input directory, messages from records preceding the point of failure are duplicated.
- **Solution:** If the input messages have unique keys, modify your flow to ignore duplicate records. If the messages do not have unique keys but each input file has a unique name, you can modify your flow to form a unique key based on the file name and record number. Define a database table and add a Database node to your flow to record the key of each record that is processed. Add a DatabaseRoute node to filter input messages so that only records without keys already in the database are processed.

  If you cannot generate unique keys for each record, split your flow into two separate flows. In the first flow, wire the FileInput node to an MQOutput node so that each input record is copied as a BLOB to an IBM MQ queue. Ensure there are adequate IBM MQ resources, queue size for example, so that the first flow does not fail. In the second flow, wire an MQInput node to the flow previously wired to your FileInput node. Configure the MQInput and other nodes to achieve the desired transactional behavior.

No file is created in the output directory after FileOutput node processing

Procedure

- **Scenario:** A file created by the FileOutput node does not appear in the output directory. The node is configured so that the `Record definition` property has a value of `Record is Unmodified`

Data, Record is Fixed Length Data, or Record is Delimited Data and the flow runs one or more times.

- **Explanation:** The FileOutput node accumulates messages, record by record, in an incomplete version of the output file in the transit subdirectory of the output directory. It moves the file from the transit subdirectory to the output directory only when it receives a message on its Finish File terminal; at this point, the file is complete. If the node's input processing fails before a message is sent to the Finish file terminal, the file remains in the transit directory. The file might be completed by a subsequent flow if it uses the same file name and output directory; if this does not happen, the file is never moved to the output directory
- **Solution:** If you need to ensure that incomplete files are moved to the output directory if the input flow fails, wire the input node's Failure terminal to the FileOutput node's Finish File terminal, in addition to all other flows that are wired to this terminal.

  If you need all output files to be available for a downstream process at a particular time or after a particular event, wire a separate flow to the FileOutput node's Finish File terminal to send a message at that particular time or on that particular event. If duplicate messages which identify the same file are sent to the Finish File terminal, the FileOutput node ignores them.

  If your flows use the Request directory property location, Request file name property location (default Directory and Name in the $LocalEnvironment/Destination/File folder), or $LocalEnvironment/Wildcard/WildcardMatch, ensure that messages sent to the Finish File terminal contain the correct elements and values to identify the output file and directory.

Output file name overrides have not been applied

Procedure

- **Scenario:** The message elements set in the flow to override the output file name or directory values specified in the FileOutput node's **Basic** properties have not been applied. The output file is created using the name and directory set in the FileOutput node's **Basic** properties.
- **Explanation:** One of the following might be the cause of this problem:
  - The message sent to the FileOutput node does not contain the expected changes.
  - The FileOutput node is configured to use different elements in the message from the ones set to the new values.
  - Not all messages contain the overriding values.
- **Solution:** Use the debugger or a Trace node inserted in front of the FileOutput node's In terminal to check that the expected overriding values appear in the correct message elements. If they do not, check that the Compute mode property has been set correctly in Compute nodes that are upstream in the flow; for example, if $LocalEnvironment/File/Name has not changed following a Compute node, check that the Compute node has its Compute mode property set to LocalEnvironment and Message.

  If the message elements are set correctly, check that the FileOutput node's Request directory property location and Request file name property location properties identify the correct elements in the message.

  If you have specified Record is Unmodified DataRecord is Fixed Length Data, or Record is Delimited Data in the FileOutput node's Record definition property, ensure that messages that go to the Finish File terminal have the same override values as those that go to the in terminal. Unless you do this, the Finish file terminal message and the In terminal messages will apply to different files.

Resolving problems when you use HTTP and SOAP nodes

Use the advice that is given here to help you to resolve common problems that can arise when you develop web services message flows that contain HTTP and SOAP nodes.

About this task

If you experience problems when you use HTTP and SOAP nodes in message flows, complete the instructions for the following scenarios to diagnose and solve the problem.

- "Error message BIP3689 is issued when you deploy or restart an integration server" on page 2924
- "Warning message BIP3690 is issued when you deploy or restart an integration server" on page 2924
- "A HandshakeException is issued when you use an HTTPRequest node to make an HTTPS call" on page 2925

Use the replies to the following questions to assist you in diagnosing problems with HTTP or SOAP nodes:

- "How do I tell which listener the HTTP and SOAP nodes are using?" on page 2925
- "How do I collect HTTPListener trace?" on page 2926

Error message BIP3689 is issued when you deploy or restart an integration server

Procedure

- **Scenario:** You configure Web Services Reliable Messaging (WS-RM) on one or more message flow nodes in your message flow configuration. When you attempt to deploy the configuration to an integration server, or to restart the integration server, you see a BIP3689 error message.
- **Explanation:** Two or more message flow nodes in your deployment configuration have the same value for the `Path suffix for URL` property, and at least one of these nodes is using WS-RM. Using the same URL twice is not allowed, as one integration server might get messages intended for another integration server, therefore breaking the WS-RM message ordering.
- **Solution:** Review the values for the `Path suffix for URL` property in the message flow nodes that you plan to deploy to the same integration node. Ensure that the URL for a node that is using WS-RM is not used anywhere else in the deployment configuration.

Warning message BIP3690 is issued when you deploy or restart an integration server

Procedure

- **Scenario:** Your integration server contains SOAP or HTTP input nodes in one or more message flows. When you attempt to deploy the integration server, or to restart it, you see a BIP3690 warning message.
- **Explanation:** This warning message indicates that you are using the same value for the `Path suffix for URL` property in both SOAP and HTTP nodes in your deployed topology. You do not see the warning if you use the same URL in two or more HTTP nodes, or in two or more SOAP nodes.

  For example, you might deploy the same flow to multiple integration servers for scaling and performance reasons; this configuration is valid and does not trigger the warning. However, using the same URL across SOAP and HTTP nodes might result in unexpected behavior, and always triggers a warning, even if the configuration is valid.

  If you are using SOAP nodes and HTTP nodes in message flows on a single integration node, you can choose to handle HTTP messages by using either the integration node listener or embedded integration server listeners. If a listener in your configuration receives messages that both SOAPInput and HTTPInput nodes might get, you must carefully check the URL specifications in these nodes. If both URL specifications match an incoming message, the wrong type of node might get the message, and processing might fail or produce unexpected results. This situation occurs if you specify identical values for the Path suffix for URL properties of the HTTPInput node and the SOAPInput node. It can

also occur if you use wildcards in either or both specifications, and an incoming message matches both properties.

- **Solution:** Check the URL specifications in your HTTP and SOAP input nodes and confirm that message routing works as intended.

A HandshakeException is issued when you use an HTTPRequest node to make an HTTPS call

Procedure

- **Scenario:** You are trying to make an HTTPS call to an external web service from your IBM App Connect Enterprise message flow by using an HTTPRequest node. You receive the following error:

```
javax.net.ssl.SSLHandshakeException:
com.ibm.jsse2.util.h: PKIX path building failed:

java.security.cert.CertPathBuilderException:
PKIXCertPathBuilderImpl could not build a valid CertPath.;

internal cause is:
java.security.cert.CertPathValidatorException:
The certificate issued by OU=Class 3 Public Primary Certification Authority,
O="VeriSign, Inc.", C=US is not trusted;

internal cause is:
java.security.cert.CertPathValidatorException: Certificate chaining error
```

- **Explanation:** The integration node cannot build the entire certificate path. The keystore of this integration node must contain all the certificates in this chain of certificate authorities (CA). For an integration node to verify the digital signature on a signed certificate, the keystore must contain the public key of the certificate authority (CA) that issued that certificate. If this public key is itself issued on a signed certificate, the keystore must contain the public key of the CA that issued that certificate. This chain continues until the integration node reaches a root certificate authority that issues a self-signed certificate.
- **Solution:** Verify that you added all the required certificates to your keystore. If any of the components in the certificate chain are missing from your keystore, re-create your keystore by using keytool with the `genkey` option, then reimport your application certificates.

How do I tell which listener the HTTP and SOAP nodes are using?

Procedure

- **Scenario:** HTTP and SOAP nodes that you include in a message flow can use either the integration node listener or the embedded listener that is defined to the integration server to which the containing message flow is deployed.

  Both listeners can handle both HTTP and HTTPS messages by handling the different message types on different ports.

- **Solution:** Use the **mqsireportproperties** command to check the properties that define what listener is in use.

  1. Check whether the integration node listener is disabled:

     ```
     mqsireportproperties INODE -b httplistener -o HTTPListener -n startListener
     ```

     If this property is false, all HTTP and SOAP nodes in all integration servers are using an embedded listener.

  2. If the integration node listener is active, you must check the specific integration server.

     For example, check whether the listener in *integrationServerName* is being used to process HTTP messages for HTTP nodes:

     ```
     mqsireportproperties integrationNodeName -e integrationServerName -o ExecutionGroup  -n
      httpNodesUseEmbeddedListener
     ```

     If this property is true, all HTTP nodes that you deploy to this integration server are using the embedded listener.

     Check all integration servers for which you want to know this information.

     Check whether the listener in *integrationServerName* is being used to process HTTP messages for SOAP nodes:

     ```
     mqsireportproperties integrationNodeName -e integrationServerName -o ExecutionGroup  -n
      soapNodesUseEmbeddedListener
     ```

     If this property is true (the default value), all SOAP nodes that you deploy to this integration server are using the embedded listener.

     Check all integration servers for which you want to know this information.

  3. If both properties are false, all HTTP and SOAP nodes that you deploy to all integration servers are using the embedded listener. The value for the integration server property is ignored.

Results

If you are experiencing problems with message flows that contain HTTP or SOAP nodes, and you want to collect trace information to provide to your IBM Support Center, trace the integration server. If you are using the integration node listener for HTTP or SOAP nodes, also trace the HTTPListener component.

How do I collect HTTPListener trace?

About this task

To gather information about HTTP nodes and listeners, you must start trace, run your message flows, then retrieve and format the trace information.

Trace the integration server and the HTTPListener component:

Procedure

- Start trace for the integration server.

  For example:

  ```
  mqsichangetrace INODE -t -e integrationServerName -l debug
  ```

- If you are using the integration node listener for one or more integration servers, start trace for the HTTPListener component in one of the following two ways:
  - Trace all integration node components by running the **mqsichangetrace** command to start trace with the following options:

    ```
    mqsichangetrace component -t -b -l debug
    ```

    where *component* is the integration node name
  - Trace only the HTTPListener component by running the **mqsichangeproperties** command to start trace with the following options:

    ```
    mqsichangeproperties integrationNodeName -b httplistener -o HTTPListener
     -n traceLevel -v debug
    ```

What to do next

Save the trace output so that you can send it to the IBM Support Center, if requested.

Turn off trace when you finish collecting information to avoid affecting the performance of the integration node.

Resolving implementation problems when developing message flows

Use the advice given here to help you to resolve some common problems that can arise when running message flows.

About this task

Messages are directed to the Failure terminal of an MQInput node

Procedure

- **Scenario:** Messages that are received at a message flow are directed immediately to the Failure terminal on the MQInput node (if it is connected), or are rolled back.

- **Explanation:** When a message is received by IBM MQ, an error is signalled if the following conditions are all true:
  - The MQInput node requests that the message content is converted (the Convert property is set to `yes` on the node).
  - The message consists only of an MQMD followed by the body of the message.
  - The message format, as specified in the MQMD, is set to MQFMT_NONE.

  This error causes the message to be directed to the Failure terminal.
- **Solution:** In general, you do not need to request IBM MQ to convert the message content, because the integration node processes messages in all code pages and encodings that are supported by IBM MQ. Set the Convert property to no to ensure that messages flow from the MQInput node to successive nodes in the message flow.

Messages enter the message flow but do not exit

Procedure

- **Scenario:** You have sent messages into your message flow, and they have been removed from the input queue, but nothing appears at the other end of the message flow.

- **Explanation:** Several situations might cause this error to occur. Consider the following scenarios to try to identify the situation that is causing your failure:

  a) Check your message flow in the IBM App Connect Enterprise Toolkit.

     You might have connected the MQInput node Failure terminal to a successive node instead of the Out terminal. The Out terminal is the *middle* terminal of the three. Messages directed to an unconnected Out terminal are discarded.

  b) If the Out terminal of the MQInput node is connected correctly to a successive node, check the integration node's local error log for an indication that message processing has been ended because of problems. Additional messages give more detailed information.

     If the Failure terminal of the MQInput node has been connected (for example, to an MQOutput node), these messages do not appear.

     Connecting a node to a Failure terminal of another node indicates that you have designed the message flow to deal with all error processing. If you connect a Failure terminal to an MQOutput node, your message flow ignores all errors that occur.

  c) If the Out terminal of the MQInput node is connected correctly to a successive node, and the local error log does not contain error messages, turn user tracing on for the message flow:

     This action produces a user trace entry from only the nodes that the message visits.

     On distributed systems, you can retrieve the trace entries by using the **mqsichangetrace** command, and view the records to check the path of the message through the message flow.

  d) If the user trace shows that the message is not taking the expected path through the message flow, increase the user trace level to Debug by selecting the message flow, right-clicking it, and clicking **User Trace** > **Debug**.

     Send your message into the message flow again. Debug-level trace produces much more detail about why the message is taking a particular route, and you can then determine the reasons for the actions taken by the message flow.

     Do not forget to turn tracing off when you have solved the problem, because performance might be adversely affected.

  e) If the MQPUT command to the output queue that is defined on the MQOutput node is not successful (for example, the queue is full or **put** is disabled), the final destination of a message depends on:

     – Whether the Failure terminal of the MQOutput node is connected.
     – Whether the message is being processed transactionally (which in turn depends on the transaction mode setting of the MQInput node, the MQOutput node, and the input and output queues).
     – Whether the message is persistent or nonpersistent. When transaction mode is set to the default value of Automatic, message transactionality is derived from the way that it was specified at the

input node. All messages are treated as persistent if transaction mode=yes, and as nonpersistent if transaction mode=no.

In general, if a path is not defined for a failure (that is, neither the Catch terminal nor the Failure terminal of the MQInput node is connected):

– Non-transactional messages are discarded.

– Transactional messages are rolled back to the input queue to be tried again:

- If the backout count of the message is less than the backout threshold (BOTHRESH) of the input queue, the message is tried again and sent to the Out terminal.
- When the backout count equals or exceeds the backout threshold, one of the following might happen:
  - The message is placed on the backout queue, if one is specified (using the BOQNAME attribute of the input queue.)
  - The message is placed on the dead-letter queue, if there is no backout queue defined or if the MQPUT to the backout queue fails.
  - If the MQPUT to the dead-letter queue fails, or if there is no dead-letter queue defined, then the message flow loops continuously trying to put the message to the dead-letter queue.

– If a path is defined for the failure, then that path defines the destination of the message. If both the Catch terminal and the Failure terminal are connected, the message is propagated through the Catch terminal.

f) If your message flow uses transaction mode=yes on the MQInput node properties, and the messages are not appearing on an output queue, check the path of the message flow.

– If the message flow has paths that are not failures (but that do not end in an output queue), either:

- The message flow has not failed and the message is not backed out.
- The message flow is put to an alternative destination (for example, the Catch terminal, the dead-letter queue, or the queue's backout queue).

– Check that all possible paths reach a final output node and do not reach a dead end. For example, check that you have:

- Connected the Unknown terminal of a Filter node to another node in the message flow.
- Connected both the True and False terminals of a Filter node to another node in the message flow.

Your integration server is not reading messages from the input queues

Procedure

- **Scenario:** Your integration server has started, but is not reading messages from the specified input queues.
- **Explanation:** A started integration server might not read messages from the input queues of the message flows because previous errors might have left the queue manager in an inconsistent state.

- **Solution:** Complete the following steps:
  a) Stop the integration node.
  b) Stop the IBM MQ listener.
  c) Stop the IBM MQ channel initiator.
  d) Stop the IBM MQ queue manager.
  e) Restart the IBM MQ queue manager.
  f) Restart the IBM MQ channel initiator.
  g) Restart the IBM MQ listener.
  h) Restart the integration node.

The integration server ends while processing messages

About this task

Procedure

- **Scenario:** While processing a series of messages, the integration server (DataFlowEngine) process size grows steadily without levelling off.

  This situation might cause the DataFlowEngine process to end if it cannot allocate more memory, and restart. The error message BIP2106 might be logged to indicate the out of memory condition.

  In addition, if you are using DB2 on distributed systems, you might get the message:

  ```
  SQL0954C  Not enough storage is  available in the application heap to process
  the statement.
  ```

- **Explanation:** When a database call is made from within a message flow node, the flow constructs the appropriate SQL, which is sent using ODBC to the database manager. As part of this process, the SQL statement is prepared using the SQLPrepare function, and a statement handle is acquired so that the SQL statement can be executed.

  For performance reasons, after the statement is prepared, the statement and handle are saved in a cache to reduce the number of calls to the SQLPrepare function. If the statement is already in the cache, the statement handle is returned so that it can be re-executed with newly bound parameters.

  The statement string is used to perform the cache lookup. By using hardcoded SQL strings that differ slightly for each message, the statement is not found in the cache, and an SQLPrepare function is always performed (and a new ODBC cursor is opened). When using PASSTHRU statements, use parameter markers so that the same SQL prepared statement can be used for each message processed, with the parameters being bound at run time. This approach is more efficient in terms of database resources and, for statements that are executed repeatedly, it is faster.

  However, it is not always possible to use parameter markers, or you might want to dynamically build the SQL statement strings at run time. This situation potentially leads to many unique SQL statements being cached. The cache itself does not grow that large, because these statements themselves are generally not big, but many small memory allocations can lead to memory fragmentation.

- **Solution:** If you encounter these types of situations, disable the caching of prepared statements by setting the MQSI_EMPTY_DB_CACHE environment variable to an arbitrary value. When this environment variable has been created, the prepared statements for that message flow are emptied at the end of processing for each message. This action might cause a slight performance degradation because every SQL statement is prepared.

Your integration server hangs, or ends with a core dump

Procedure

- **Scenario:** While processing a message, an integration server either hangs with high CPU usage, or ends with a core dump. The stack trace from the core dump or abend file is large, showing many calls on the stack. Messages written to the system log might indicate "out of memory" or "bad allocation" conditions. The characteristics of the message flow in this scenario often include a hard-wired loop around some of the nodes.
- **Explanation:** When a message flow thread executes, it requires storage to perform the instructions that are defined by the logic of its connected nodes. This storage comes from the integration server's heap and stack storage. The execution of a message flow is constrained by the stack size, the default value of which differs depending on the operating system.
- **Solution:** If a message flow that is larger than the stack size is required, you can increase the stack size limit and then restart the integration nodes that are running on the system so that they use the new value.

  For information on setting the stack size for your operating system, see System resources for message flow development.

Your XSLTransform node is not working after deployment and errors are issued indicating that the style sheet could not be processed

Procedure

- **Scenario:** Your XSLTransform node is not working after deploying resources, and errors are displayed indicating that the style sheet could not be processed.
- **Solution:**
  - If the integration node cannot find the style sheet or XML files that are required, migrate the style sheets or XML files with relative path references.
  - If the contents of a style sheet or XML file are damaged and therefore no longer usable (for example, if a file system failure occurs during a deployment), redeploy the damaged style sheet or XML file.

Output messages are not sent to expected destinations

Procedure

- **Scenario:** You have developed a message flow that creates a destination list in the LocalEnvironment tree. The list might contain queues for the MQOutput node, labels for a RouteToLabel node, or URLs for an HTTPRequest node. However, the messages are not reaching these destinations, and there are no error messages.
- **Solution:**
  - Check that you have set Compute mode to a value that includes the LocalEnvironment in the output message, for example All. The default setting of Compute mode is Message, and all changes that you make to LocalEnvironment are lost.
  - Check your ESQL statements. The content and structure of LocalEnvironment are not enforced, so the ESQL editor (and content assist) does not provide guidance for field references, and you might have specified one or more of these references incorrectly.

    Some example procedures to help you set up destination lists are provided in “Populating Destination in the local environment tree” on page 1656. You can use these procedures unchanged, or modify them for your own requirements.

You experience problems when sending a message to an HTTP node's URL

Procedure

- **Scenario:** Sending a message to an HTTP node's URL causes a timeout, or the message is not sent to the correct message flow.
- **Explanation:** The following rules are true when URL matching is performed:
  - There is one-to-one matching of HTTP requests to HTTPInput nodes. For each HTTP request, only one message flow receives the message. This statement is true even if two message flows are listening on the same URL. Similarly, you cannot predict which MQInput node that is listening on a particular queue will receive a message.
  - Messages are sent to wildcard URLs only if no other URL is matched. Therefore a URL of /* receives all messages that do not match another URL.
  - Changing a URL in an HTTPInput node does not automatically remove the entry from the HTTP listener. For example, if a URL /A is used first, then changed to a URL of /B, the URL of /A is still used to listen on, even though there is no message flow to process the message. This incorrect URL does get removed after the integration node has been stopped and restarted twice.
- **Solution:** To find out which URL the integration node is currently listening on, look at the file `wsplugin6.conf` in the following location:
  - Linux UNIX On Linux and UNIX: `/var/mqsi/components/`*`integrationNodeName`*`/config`
  - Windows On Windows, `%ProgramData%\IBM\MQSI\components\`*`integrationNodeName`*`\config`, where %PROGRAMDATA% is the environment variable that defines the system working directory. The default directory depends on the operating system.

  If problems persist, empty `wsplugin6.conf`, restart the integration node, and redeploy the message flows.

When using secure HTTP connections, you change a DNS host's destination but the integration node is using a cached DNS host definition

Procedure

- **Scenario:** You are using an integration node with secure HTTP connections that use the Java virtual machine (JVM). You have changed a DNS destination, but the integration node is using a cached DNS host definition, therefore you have to restart the integration node to use the new definition.
- **Explanation:** By default, Java caches the host lookup from DNS, which is not appropriate if you want to look up the host name each time or if you want to cache it for a limited amount of time. This situation occurs only when you use SSL connections. (When using a secure HTTPS connection, the HTTPRequest node uses the SSL protocol, which issues Java calls, whereas a non-SSL protocol uses native calls.)

  To avoid this situation, you can empty the cache on the JVM by setting the **`networkaddress.cache.ttl`** property to zero. This property dictates the caching policy for successful name lookups from the name service. The value is specified as an integer to indicate the number of seconds for which to cache the successful lookup. The default value of this property is -1, which indicates that the successful DNS lookup value is cached indefinitely in the JVM. If you set this property to 0 (zero), the successful DNS lookup is not cached.

- **Solution:** To pick up DNS entry changes without the need to stop and restart the integration node and JVM, disable DNS caching.

  Edit file `$JAVA_HOME/jre/lib/security/java.security`, and set the value of the **`networkaddress.cache.ttl`** property to `0` (zero).

  Then, run the following command:

```
mqsichangeproperties <INodeName> -e <IntegrationServer> -o ComIbmJVMManager -n
 jvmSystemProperty -v "-Dsun.net.inetaddr.ttl=0"
```

The TimeoutControl node issues error message BIP4606 or BIP4607 when the timeout request start time that it receives is in the past

Procedure

- **Scenario:** When a TimeoutControl node receives a timeout request message that contains a start time in the past, it issues error message BIP4606 or BIP4607: `The Timeout Control Node '&2' received a timeout request that did not contain a valid timeout start date/time value`.
- **Explanation:** The start time in the message can be calculated by adding an interval to the current time. If a delay occurs between the node that calculates the start time and the TimeoutControl node, the start time in the message will have passed by the time it reaches the TimeoutControl node. If the start time is more than approximately five minutes in the past, a warning is issued and the TimeoutControl node rejects the timeout request. If the start time is less than five minutes in the past, the node processes the request as if it were immediate.
- **Solution:** Ensure that the start time in the timeout request message is a time in the future.

You are using a TimeoutControl node with a TimeoutNotification node, with multiple clients running concurrently, and messages appear to be being dropped

Procedure

- **Scenario:** You are using a TimeoutControl node with a TimeoutNotification node, with multiple clients running concurrently, and messages appear to be being dropped. In the timeout request message, allowOverwrite is set to TRUE.
- **Explanation:** If multiple clients are running concurrently, and allowOverwrite is set to TRUE in the timeout request message, messages can overwrite each other.
- **Solution:** Ensure that different TimeoutNotification nodes that are deployed on the same integration node do not share the same unique identifier.

Error message BIP5347 is issued on AIX when you run a message flow that uses a message set

About this task

Procedure

- **Scenario:** Error message BIP5347 (`MtilmbParser2: RM has thrown an unknown exception`) is issued on AIX in either of these circumstances:
  - When you are deploying a message set
  - When you are running a message flow that uses a message set

- **Explanation:** BIP5347 is typically caused by a database exception, and it is issued when an integration server tries to load an MRM dictionary for use by a message flow. This process involves two steps:

  1. The integration server retrieves the dictionary and wire format descriptors from the integration node data store.
  2. The integration server stores the dictionary in the memory that a message flow would use to process an MRM message.

  BIP5347 is typically issued during step 1. This problem can appear to be intermittent; if you restart the integration server, the message is sometimes processed correctly.

  BIP5347 might also be caused by the presence of a datetime value constraint in the message set, which causes the error each time the message set is deployed.
- **Solution:** To identify the cause of the error, capture a service level debug trace to confirm that the database exception is occurring.
  - If the error is caused by the presence of a datetime value constraint, a message similar to the following message appears in the service level debug trace (the exact message depends on the datetime constraint in the message set):

    ```
    Unable to parse datetime internally, 9, 2001-12-17T09:30:47.0Z,
    yyyy-MM-dd'T'HH:mm:ss.SZZZ
    ```

    This error occurs because the MRM element in question has a datetime value that is not compatible with the datetime format string, so the dictionary is rejected. To solve this problem, ensure that the datetime value is compatible with the datetime format string.

Error message BIP2130 is issued with code page value of -1 or -2

Procedure

- **Scenario:** The following error message is issued:

```
BIP2130: Error converting a character string to or from codepage [code page value]
```

  where `[code page value]` is either -1 or -2. You have not, however, specified a code page of either -1 or -2 in your message tree. You have, however, used one of the IBM MQ constants MQCCSI_EMBEDDED or MQCCSI_INHERIT.
- **Explanation:** The IBM MQ constants MQCCSI_EMBEDDED and MQCCSI_INHERIT are resolved when the whole of the message tree is serialized to produce the IBM MQ bit stream. This happens when the message is put on the IBM MQ transport. Until that time, these values exist in the message tree as either -1 (for MQCCSI_EMBEDDED) or -2 (for MQCCSI_INHERIT). If one or more parts of the message tree are serialized independently, such as with a ResetContentDescriptor node or ESQL ASBITSTREAM function, this error occurs.
- **Solution:** You do not have to set MQCCSI_EMBEDDED or MQCCSI_INHERIT in the message tree's CodedCharSetId field. You can achieve the same result by explicitly setting the required CodedCharSetId to the previous header's CodedCharSetId value. For example, you would need to replace:

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)CodedCharSetId = MQCCSI_INHERIT;
```

  with

```
SET OutputRoot.MQRFH2.(MQRFH2.Field)CodedCharSetId = InputRoot.MQMD.CodedCharSetId;
```

  where the MQMD folder is the header preceding the MQRFH2 header.

The integration server restarts before an MQGet node has retrieved all messages

Procedure

- **Scenario:** You have created a message flow that contains an MQGet node. Not all of the messages are retrieved from the queue because the integration server restarts before the node has retrieved all the messages. No abend files are generated.
- **Explanation:** In IBM App Connect Enterprise, processing that involves nested or recursive processing can cause extensive use of the stack. Message flow processing occurs in a loop until the MQGet node has retrieved all the messages from the queue. Each time that processing returns to the MQGet node, the stack size increases.
- **Solution:** Use a PROPAGATE statement. The statement propagates each message through the message flow in a loop, but each time that processing returns to the PROPAGATE statement, the stack is cleared.

  Use an ESQL variable (for example, set Environment.Complete to true) in the environment tree to terminate the ESQL loop, stop the propagations, and wait for the next trigger message. If you need to store content from the messages, store it in the environment tree because other trees are deleted when message flow processing returns to the PROPAGATE statement. For more information about how to use this statement, see PROPAGATE statement.

Resolving problems when you use IMS nodes

Advice for dealing with common problems that can arise when you develop message flows that contain IMS nodes.

Before you begin

- Read about IMS in "IBM Information Management System (IMS)" on page 1146.
- Ensure that you have set up the IBM App Connect Enterprise runtime environment correctly, as described in "Preparing the environment for IMS nodes" on page 215.

About this task

If you experience problems when you use IMS nodes in message flows, follow the instructions for the following scenarios to diagnose and solve the problem.

How can I tell if my integration node is connected to IMS Connect?

Procedure

- **Scenario:** You need to check if your integration node is connected to IMS Connect.
- **Explanation:** You can use SDSF on z/OS to issue a command to see which ports have clients connected to them.

- **Solution:** Using SDSF, enter the following **QUERY MEMBER** command, where *IM0ACONN* is the name of your IMS Connect job:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
```

The output is in the following format:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
HWSC0001I   HWS ID=IM0ACONN RACF=Y  PSWDMC=N
HWSC0001I      MAXSOC=50  TIMEOUT=0
HWSC0001I      RRS=N   STATUS=REGISTERED
HWSC0001I      VERSION=V10 IP-ADDRESS=009.017.252.024
HWSC0001I      SUPER MEMBER NAME=
HWSC0001I    ADAPTER=N
HWSC0001I    DATASTORE=IM0A     STATUS=ACTIVE
HWSC0001I      GROUP=IM0AGRNM MEMBER=IM0ACONN
HWSC0001I      TARGET MEMBER=IM0A
HWSC0001I      DEFAULT REROUTE NAME=HWS£DEF
HWSC0001I      RACF APPL NAME=
HWSC0001I      OTMA ACEE AGING VALUE=2147483647
HWSC0001I      OTMA ACK TIMEOUT VALUE=120
HWSC0001I      OTMA MAX INPUT MESSAGE=5000
HWSC0001I    NO ACTIVE IMSPLEX
HWSC0001I    PORT=1080     STATUS=ACTIVE
HWSC0001I      CLIENTID USERID   TRANCODE STATUS       SECOND  CLNTPORT  IP-ADDRESS

HWSC0001I      HWSEHYMO JDOE   IVTNO    RECV             21      1109  009.017.137.11

HWSC0001I      TOTAL CLIENTS=1  RECV=1 CONN=0 XMIT=0 OTHER=0
```

This example shows that one client is connected on TCP port 1109, and that the client is connecting from the IP address 9.17.137.11.

You can run netstat on that system to find out in which process that client is running.

Use the following IMS command to view the TPIPEs that are created for those connections:

```
/DISPLAY TMEMBER IMSConnect_Name TPIPE ALL
```

- For transactions that have a commit mode of 1, the TPIPE name is the port number that is used for that interaction (for example, 1080 in the previous example).
- For transactions that have a commit mode of 0, the TPIPE name is the same as the client ID (for example, HWSEHYMO in the previous example). The client ID is generated automatically in the IMSRequest node.

How can I discover the correct settings for `Hostname`, `Portnumber`, and `DataStoreName`?

Procedure

- **Scenario:** Your integration node fails to connect to IMS and you want to verify your settings.
- **Explanation:** You can use SDSF on z/OS to issue a command to see members of the XCF group to which your IMS control region belongs. One of these members should be IMS Connect. You can then run a command against IMS Connect to discover these properties.
- **Solution:** Use SDSF to enter the following command:

```
/xx/display OTMA
```

where *xx* is the reply ID for your IMS control region job.

For example, if you see `....... *26 DFS996I *IMS READY* IM0A` in `S.log`, run the command `/26/DISPLAY OTMA`.

The output from that command shows the name of the IMS Connect that is in the same XCF group as this IMS control region:

```
DFS000I     GROUP/MEMBER      XCF-STATUS   USER-STATUS    SECURITY  TIBINPT SMEM   IM0A
```

```
DFS000I                          DRUEXIT  T/O                                          IM0A
DFS000I     IM0AGRNM                                                                  IM0A
DFS000I     -IM0A             ACTIVE       SERVER         CHECK                      IM0A
DFS000I     -IM0A               N/A        0                                         IM0A
DFS000I     -IM0ACONN         ACTIVE       ACCEPT TRAFFIC CHECK       05000          IM0A
DFS000I     *08350/175112*   IM0A
```

Use SDSF to enter the following **QUERY MEMBER** command, where *IM0ACONN* is the name of your IMS Connect job that was reported by the previous command:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
```

The output is in the following format:

```
/F IM0ACONN,QRY MEMBER TYPE(IMSCON)
HWSC0001I   HWS ID=IM0ACONN RACF=Y  PSWDMC=N
HWSC0001I      MAXSOC=50  TIMEOUT=0
HWSC0001I      RRS=N   STATUS=REGISTERED
HWSC0001I      VERSION=V10 IP-ADDRESS=009.017.252.024
HWSC0001I      SUPER MEMBER NAME=
HWSC0001I    ADAPTER=N
HWSC0001I    DATASTORE=IM0A     STATUS=ACTIVE
HWSC0001I      GROUP=IM0AGRNM MEMBER=IM0ACONN
HWSC0001I      TARGET MEMBER=IM0A
HWSC0001I      DEFAULT REROUTE NAME=HWS£DEF
HWSC0001I      RACF APPL NAME=
HWSC0001I      OTMA ACEE AGING VALUE=2147483647
HWSC0001I      OTMA ACK TIMEOUT VALUE=120
HWSC0001I      OTMA MAX INPUT MESSAGE=5000
HWSC0001I    NO ACTIVE IMSPLEX
HWSC0001I    PORT=1080     STATUS=ACTIVE
HWSC0001I      CLIENTID USERID   TRANCODE STATUS       SECOND  CLNTPORT  IP-ADDRESS
HWSC0001I      HWSEHYMO JDOE   IVTNO    RECV             21      1109  009.017.137.11
HWSC0001I      TOTAL CLIENTS=1  RECV=1 CONN=0 XMIT=0 OTHER=0
```

What should I do when my transaction times out?

Procedure

- **Scenario:** The transaction times out and the message is sent to the Timeout terminal, or an exception is issued.
- **Explanation:** Your transaction is taking longer than the values that are set for the execution or socket timeouts, therefore the node stops waiting for a response, and issues an exception or sends the message to the Timeout terminal.

  If the transaction subsequently completes successfully, the result depends on the commit mode that is set on the IMSRequest node:

  – If the `Commit mode` property is set to `0: COMMIT_THEN_SEND`, the unit of work is committed and the response is discarded.

  – If the `Commit mode` property is set to `1: SEND_THEN_COMMIT`, the response is not sent and the unit of work is rolled back.
- **Solution:** Increase the execution or socket timeout values to give enough time for the transaction to complete.

  – Configure the execution timeout by using the `Timeout waiting for a transaction to be executed` property on the IMSRequest node.

How many physical connections should I expect in IMS Connect?

Procedure

- **Scenario:** You need to set the values for the number of connections that are required by clients that are connecting to IMS Connect.

- **Explanation:** The number of physical connections that can be opened in IMS Connect is limited. The limit depends on the MAXSOC and MAXFEILEPROC settings.
  - The MAXSOC setting in IMS Connect determines the number of sockets that can be opened in IMS, which is the number of ports on which IMS listens for connections, plus the number of physical connections.
  - MAXFILEPROC is a UNIX System Services (USS) setting, which must be greater than or equal to MAXSOC, otherwise IMS reaches this limit before it reaches its own MAXSOC limit.

  If the IMS Connect process is granted superuser authority in USS, it sets MAXFILEPROC automatically.

  If the MAXSOC value is reached, IMS Connect issues warning message HWSS0771W, and refuses new requests for connections from clients. This behavior continues until the number of open sockets is below the limit (for example, after some clients have disconnected).

  If the MAXFILEPROC value is reached, USS issues information message BPXI040I.
- **Solution:** When you set values for MAXSOC and MAXFILEPROC, consider how many clients are likely to connect concurrently to IMS Connect, and how many connections those clients will require.

  IBM App Connect Enterprise acts as a client to IMS Connect, and opens connections to IMS Connect. Therefore, find out how many connections are required by IBM App Connect Enterprise by gathering the following information:
  - The number of message flows with IMS nodes that are deployed
  - For those message flows, the values of the `Additional instances` property

  The maximum number of connections required for each integration node is determined by the number of threads that can be running concurrently in the IMS nodes. In the following example, three message flows with IMS nodes exist:
  - Message flow A has 0 additional instances, therefore one thread is running.
  - Message flow B has 3 additional instances, therefore four threads can be running concurrently.
  - Message flow C has 4 additional instances, therefore five threads can be running concurrently.

  In this example, the maximum number of connections required by the integration node is 10 (1+4+5). If you have four other similar integration nodes, all connecting to the same instance of IMS Connect, which has five ports configured, you would set the maximum number of sockets (MAXSOC) to at least 55 (the maximum number of connections for five integration nodes plus the number of ports: 10x5+5).

Resolving problems when you use Salesforce nodes

Use the advice given here to help you to avoid or resolve problems that might arise when you are using SalesforceRequest nodes.

Connecting to Salesforce

Procedure

- When you set the Salesforce password by using the mqsisetdbparms command, ensure that the password is suffixed with the Salesforce security token for the Salesforce ID. You can get the security token from the email that was sent to the email address associated with the credentials when the security token was last reset.

  For more information, see "Configuring a secure connection to Salesforce.com" on page 1196.
- After creating a new IBM App Connect Enterprise Connected App in Salesforce.com, click **Manage** and ensure that the OAuth policy **Permitted Users** is set to `All users may self-authorize`. If this option is not selected, connections might fail even if the correct credentials are supplied.
- Ensure that the Salesforce user ID that you use allows for the number of operations that you want to perform. Some types of account apply quotas to the number of operations that can be performed in a given time period using the Force.com REST API.

Working with Salesforce records

Procedure

- If you want to use a SalesforceRequest node for production purposes, you must purchase a separate license entitlement for IBM Application Integration Suite. You must then change the operation mode of the integration node to `applicationIntegrationSuite` mode.

  For more information, see "License requirements" on page 5 and "Changing the operation mode" on page 86.
- You can reduce the amount of data returned from Salesforce through a `retrieve` operation, by using a **`field`** filter to specify the particular fields that you want to retrieve. Performance is improved as a result because, if no fields are specified, an extra call is made by the Salesforce client code to retrieve the full list of fields.
- After a `retrieve` operation, the output message body will be a JSON array, unless a Salesforce ID is specified to retrieve a single record.
- If you are using the JSON schema for Salesforce objects in a Mapping node, the schema and the message map must reside in the same shared library or REST API. For more information, see "Using Salesforce models" on page 1212.
- Several fields in Salesforce objects are read-only, and an attempt to create or update a read-only field will return an error from Salesforce. Fields that are read-only are typically system properties (such as '`CreatedDate`') or derived fields (such as '`BillingAddress`').

Handling errors returned from the SalesforceRequest node

About this task

When you are using the SalesforceRequest node, several different types of failure can be propagated to the failure terminal of the node, including the following:

- **Missing or badly formed data is supplied to the SalesforceRequest node**.

  If the validation of the input data in the message body or the local environment fails, error `bip3859` or `bip3862` will be propagated to the Failure terminal, which contains details of the incorrectly formed or missing data. For example, if a request is made to retrieve a set of records but the filter property in the LocalEnvironment specifies the limit clause as a negative value, error `bip3859` will be sent to the Failure terminal with details of the field that caused the validation error:

  ```
  Feb 22 11:32:02 localhost IIB[25130]: IBM Integration Bus v10004 (Server1.default) [Thread
   26660] (Msg 3/3) BIP3859E: The property 'filter.limit'='-1' supplied to the SalesforceRequest
   node is invalid.
  ```

- **A communications failure with Salesforce**.

  If there is a communications failure with Salesforce, for example if an invalid URL was supplied or an authorization failure occurs, error `bip3858` will be sent to the Failure terminal of the SalesforceRequest node, with the message inserts containing details of the failure. For example, if a request is made to retrieve a set of records but the URL provided on the SalesforceRequest node is incorrect, error `bip3858` will be sent to the Failure terminal with details of the error:

  ```
  Feb 22 11:37:01 localhost IIB[25130]: IBM Integration Bus v10004 (Server1.default) [Thread
   27379] (Msg 3/3) BIP3858E: The SalesforceRequest node received error code 'Error' while
   executing 'Retrieve' operation. Error 'getaddrinfo ENOTFOUND badlogin.salesforce.com'.
  ```

- **The Salesforce operation returned a failure**.

  If the Salesforce operation executed but returned a failure, error `bip3858` will be sent to the Failure terminal of the SalesforceRequest node, with the message inserts containing details of the failure. For example, if a request is made to retrieve a set of records but the filter property in the LocalEnvironment

specifies an invalid name in the field clause, error `bip3858` will be sent to the Failure terminal with details of the error:

```
Feb 22 12:05:36 localhost IIB[25130]: IBM Integration Bus v10004 (Server1.default) [Thread
 31343] (Msg 3/3) BIP3858E: The SalesforceRequest node received error code 'INVALID_FIELD'
 while executing 'Retrieve' operation. Error ' SELECT Id, Name, RemoteAddress FROM Account
 ^ ERROR at Row:1:Column:18 No such column 'RemoteAddress' on entity 'Account'. If you are
 attempting to use a custom field, be sure to append the '__c' after the custom field name.
 Please reference your WSDL or the describe call for the appropriate names.'.
```

- **An update or delete request was made specifying a Salesforce identifier, but no record exists with that Salesforce ID**.

  This error code is returned only for the `update` and `delete` requests; if no record exists for a `retrieve` request, no error is returned, but an empty result is returned through the Out terminal to the SalesforceRequest node. For example, if a request is made to delete a record, but no record exists with the specified Salesforce Id, error `bip3863` is sent to the failure terminal:

```
Feb 22 12:16:47 localhost IIB[25130]: IBM Integration Bus v10004 (Server1.default) [Thread
 414] (Msg 3/3) BIP3863E: A request was sent to Salesforce specifying an ID with value
 '000000000000000000' while processing the 'delete' operation, but no Salesforce record could
 be found with that ID.
```

  This is not a complete list of the errors code which may be returned from the SalesforceRequest node.

Resolving problems when using Kafka nodes

Use the advice given here to help you to avoid or resolve problems that might arise when you are using KafkaConsumer or KafkaProducer nodes.

Connecting to a Kafka cluster

Procedure

- When you set the Kafka user ID and password by using the **mqsisetdbparms** command, ensure that you used the form `kafka::KAFKA::`*integrationServerName*.

  For more information, see “Configuring security credentials for connecting to Kafka” on page 904.
- If you are connecting to IBM Event Streams in IBM Cloud, ensure that the **Security protocol** property on the Kafka node is set to SASL_SSL. For more information about configuring the security credentials for connecting to Event Streams, see “Using Kafka nodes with IBM Event Streams” on page 909.

Using log4j to investigate connection and configuration problems

About this task

If you have a problem when you connect to Kafka, and the problem persists after you validate that the configuration of the nodes is correct, you might find more information by enabling tracing from the Kafka client.

The Kafka client uses the log4j component to capture trace information. The `lib` subdirectory of the IBM App Connect Enterprise installation directory contains a `log4j.properties` file, which contains an example of how tracing can be enabled for the Kafka client:

```
################################################################################
 ## Kafka logger, un-comment the following section to write Kafka log messages
 ## to a file.
 ## The logging level, currently 'DEBUG' is the most detailed level of
 ## logging available. The log4j.appender.kafkaAppender.Threshold value may
 ## be changed to one of
 ##     DEBUG, TRACE, INFO, WARN, ERROR, FATAL
 ## to reduce the information written to the log file.
 ################################################################################
 log4j.logger.org.apache.kafka=DEBUG, kafkaAppender
 log4j.appender.kafkaAppender=org.apache.log4j.FileAppender
```

```
log4j.appender.kafkaAppender.File=/tmp/kafkadebug.log
log4j.appender.kafkaAppender.Threshold=DEBUG
log4j.appender.kafkaAppender.Append=true
log4j.appender.kafkaAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.kafkaAppender.layout.ConversionPattern=[%d] %p %m (%c)%n
```

Procedure

Complete the following steps to enable tracing from the Kafka client:

1. Copy the `log4j.properties` file from the `lib` subdirectory of the IBM App Connect Enterprise installation directory to the **workpath** of the integration server:

   For an independent integration server, the target directory is shown in the following example:

   ```
   - cp <install dir>/server/lib/log4j.properties <work-dir>/log4j.properties
   ```

   For a managed integration server, the target directory is shown in the following example:

   ```
   - cp <install dir>/server/lib/log4j.properties <workpath>/components/<integrationNodeName>/
   servers/<serverName>/log4j.properties
   ```

   For all integration servers that are managed by the same integration node, the target directory is shown in the following example:

   ```
   - cp <install dir>/server/lib/log4j.properties <workpath>/components/<integrationNodeName>/
   log4j.properties
   ```

2. In the file that you copied, uncomment the `kafkaAppender` configuration, and modify the level of logging and the destination for the captured trace, as required.
3. Restart the integration server to start capturing logging information from the Kafka client.

Enabling JSSE trace to troubleshoot issues

About this task

If you have a problem when you connect to Kafka, and the problem persists after you validate that the configuration of the nodes is correct, you might find more information by enabling JSSE tracing.

At the time of writing, the following instructions work only for HTTP listeners. If you want to troubleshoot an application that uses secure connection (HTTPS) you can gather agent service trace if you are using the integration node listener, or DataFlowEngine service trace if you are using the embedded listener. In some cases, you might need to collect both types of traces concurrently. For more information, see “Using trace” on page 3026.

Procedure

You can enable JSSE trace by setting the environment variable *IBM_JAVA_OPTIONS* as follows:

```
export IBM_JAVA_OPTIONS=-Djavax.net.debug=all
```

Alternatively, enable JSSE trace by completing the following steps:

1. Stop the integration node or integration server as described in “Starting and stopping an integration node” on page 247 and “Stopping an integration server” on page 254.
2. In the `server.conf.yaml` file for the integration server, update the parameter **jvmSystemProperty** as follows:

   ```
   jvmSystemProperty: '-Djavax.net.debug=true'
   ```

   For more information about updating the `server.conf.yaml` file, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172.

3. Restart the integration node or integration server as described in “Starting and stopping an integration node” on page 247 and “Starting an integration server” on page 250.
4. Re-create the issue.

   If you set the parameter correctly, the text "IBMJSSE2 will not allow protocol SSLv3 per com.ibm.jsse2.disableSSLv3 set to TRUE or default" appears in standard output (STDOUT). If you do not see this text, review your steps and try again. For more information about standard output (STDOUT), see Standard system logs.

Resolving mapping and message reference problems when developing message flows

Advice for dealing with some common mapping and message reference problems that can arise when developing message flows:

Resources that are referenced by the mapping file cannot be resolved

Procedure

- **Scenario:** You have imported some message flows into the IBM App Connect Enterprise Toolkit that contain mappings. An error is issued, indicating that the resources that are referenced by the mapping file cannot be resolved.
- **Explanation:** Mappings can use resources that exist in other projects. For example, a mapping reference to a message set might exist in a different project. If the reference cannot be resolved, it probably means that the reference to the other project has been lost.
- **Solution:** Create a new project reference.

  For more information, see “Referencing resources in other libraries” on page 1976.

Warnings or errors are issued for message references

Procedure

- **Scenario:** Warnings or errors are issued for message references, yet you are certain that your references are correct.
- **Explanation:** This is never the case with messages that are using the XML parser. For these message references, direct validation is not performed because the references could be used for generic XML.

  There is an ESQL editor preference that allows you to choose to ignore message reference mismatches, or to have them be reported as a warning or an error. By default, this type of problem is reported as a warning, so that you can still deploy the message flow.
- **Solution:** To use the validation feature, ensure that you have set up a project reference from the project that contains the ESQL to the project that contains the message set.

  For more information, see “Referencing resources in other libraries” on page 1976.

  If you are using reference in a subroutine, take the following steps:

  a) Create a reference to the tree and the parser in the module's main procedure.

  b) Associate the reference to the correlation name, for example InputRoot or Root. Alternatively, create the OutputRoot.*parser* node, where *parser* is the name of the parser that you want to use.

  c) Pass the reference as a parameter to an ESQL subroutine that identifies the XSD type of the reference.

Results

This practice is beneficial because the passed reference supports content assistance and validation for ESQL. The message type content properties `open`, or `open defined` are not used in validation, and the assumption is that this property is `closed`.

Target is not referencing a valid variable warning when you set the value of a target

Procedure

- **Scenario:** You have set the value for a target to a variable, such as an IBM MQ constant, and when you save the map file the warning `The target "$target" is not referencing a valid variable` is generated.
- **Explanation:** The variable that you have referenced is not recognized. For example, you might have entered an expression of the form `$mq:` followed by an IBM MQ constant, but the constant is not recognized. This might be because the variable has been entered incorrectly or it is not supported. Alternatively, you might be referencing a new variable or constant that can be resolved only at run time. If this is the case you can ignore the warning.
- **Solution:** Try one of the following to solve the problem:
  - Check that the variable has been entered correctly.
  - If you are using IBM MQ constants, use **Edit** > **Content Assist** to select from the list of available IBM MQ constants.

There are missing or unexpected targets in a message map

Procedure

- **Scenario:** In your message map, warning messages are displayed that indicate a target element is missing, or a target element is at an unexpected location. As a result the output message generated by the message map might be incorrect.
- **Explanation:** If target elements are missing from the Spreadsheet pane when you edit and save the message map, a warning is displayed that a target is missing. This situation can occur if you use the **Insert Children** wizard and do not select all the required elements, or if you create mappings using the drag-and-drop method and do not create mappings for all the required fields. If target elements in the Spreadsheet pane are in an unexpected order, a warning that the element is unexpected at that location is displayed. This situation can occur if you drag elements to new locations in the Spreadsheet pane.
- **Solution:** To solve the problem:
  - Use **Insert Children** on the parent element to add any missing target elements to the Spreadsheet pane.
  - Drag any target elements that are in an unexpected location to the correct location. Use the message tree in the Target pane as a guide to the expected structure of the output message.

Resolving trace problems when developing message flows

Follow this advice to deal with some common trace problems that can arise when you develop message flows.

About this task

- "You cannot determine which node is being referenced in your trace file" on page 2944
- "You cannot see any alerts when you change user trace " on page 2945
- "Data that the Trace node sends to the syslog on Linux is truncated" on page 2945

You cannot determine which node is being referenced in your trace file

Procedure

- **Scenario:** You have generated a service trace file for your message flow, to trace the path of a message. However, you cannot determine which node is being referenced in the trace file.

- **Explanation:** In the trace file you might see text such as:

  ```
  Video_Test#FCMComposite_1_1 ComIbmMQInputNode , Video_Test.VIDEO_XML_IN
  ```

  The elements in the text have the following meanings:

  Video_Test
  is the name of the message flow

  FCMComposite_1_1
  is the internal name for the node

  ComIbmMQInputNode
  is the type of node

  VIDEO_XML_IN
  is the node label, and is the name you see in your flow

  The number at the end of the internal name is incremented; for example, FCMComposite_1_4 would be the fourth node you added to your flow. In the example, this section of the trace is referring to the first node in the message flow.

You cannot see any alerts when you change user trace

Procedure

- **Scenario:** You cannot see any alerts for an integration server or message flow in the Alerts viewer when you use the **mqsichangetrace** command to change the user trace setting.
- **Explanation:** No alert is generated when an integration server runs user trace at normal or debug level. Alerts are generated only for message flow trace. For message flow trace, the IBM App Connect Enterprise Toolkit is not notified of trace changes that are initiated by the **mqsichangetrace** command.
- **Solution:** To see the alert, refresh the message flow, or disconnect, then reconnect to the domain.

Data that the Trace node sends to the syslog on Linux is truncated

Procedure

- **Scenario:** You are using a Trace node on Linux and have set the `Destination` property to `Local Error Log`. You send a message to the Trace node that consists of multiple lines, but the message that appears in the syslog is truncated at the end of the first line.
- **Explanation:** On Linux, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the new line character.
- **Solution:** To record a large amount of data in a log on Linux, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

Resolving problems when developing message flows with WebSphere Adapters nodes

Advice for dealing with common problems that can arise when you develop message flows that contain WebSphere Adapters nodes.

About this task

WebSphere Adapters nodes

- "Error messages BIP3414 and BIP3450 are issued when you deploy a WebSphere Adapters input node" on page 2946
- "Error messages are issued when classes cannot be found, or when problems occur with Java initialization " on page 2946

- "The WebSphere Adapters are not visible when you run ITLM" on page 2947
- "A message flow with an SAPRequest, SiebelRequest, or PeopleSoftRequest node has deployed successfully, but message BIP3540 is issued indicating that connection failed" on page 2947

SAP nodes

- "You have deployed an SAP inbound adapter but do not receive expected messages" on page 2948
- "You have imported an existing project into your workspace, but messages are issued when you try to build SAP message sets" on page 2948
- "An error is issued when you use the message set that is generated by the Adapter Connection wizard" on page 2948
- "When you run the SAP samples on Linux or UNIX, IBM App Connect Enterprise does not connect to the SAP gateway" on page 2949
- "SAP inbound messages (ALE and BAPI) appear to be missing" on page 2949
- "When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs" on page 2949

Siebel nodes

- "You are using a SiebelInput node with the delivery type set to unordered, and error message BIP3450 is issued with a NullPointer exception" on page 2950

JD Edwards nodes

- "Error message BIP3450 is issued when you include a JDEdwardsRequest node in a message flow and try to connect to a JD Edwards EnterpriseOne server" on page 2950

Error messages BIP3414 and BIP3450 are issued when you deploy a WebSphere Adapters input node

Procedure

- **Scenario:** When you deploy a message flow that contains a SiebelInput node, error message BIP3414 is issued.
- **Explanation:** The error messages explain that the SiebelInput node could not register with the adapter component to receive events.

  This problem can be caused when the integration node does not know where to find the client libraries for the Siebel Enterprise Information Service (EIS). You might also encounter this problem if you are using the WebSphere Adapter for Siebel on an unsupported operating system.
- **Solution:** Use the **mqsireportproperties** and **mqsichangeproperties** commands to configure the integration node with the location of the Siebel client libraries, as described in "Preparing the environment for WebSphere Adapters nodes" on page 209.

Error messages are issued when classes cannot be found, or when problems occur with Java initialization

Procedure

- **Scenario:** You are deploying WebSphere Adapters, and error messages are issued that indicate that classes cannot be found, or problems are occurring with Java initialization. BIP3521 and BIP3522 error messages might also be issued.
- **Explanation:** The SAP, Siebel, and PeopleSoft adapters need client libraries from the manufacturer of the Enterprise Information System (EIS). If these libraries are missing, not installed correctly, or at an incorrect level, errors are issued.

- **Solution:** To solve this problem, complete the following steps.
  a) Ensure that the adapter policy correctly identifies the location of the Java and native libraries (see Policy properties).
  b) Ensure that the libraries are installed correctly, are valid for your operating system, and have the correct permission so that the integration node can access them.
  c) Ensure that your operating system is supported by IBM App Connect Enterprise and the EIS provider. For details about supported operating systems, visit the IBM App Connect Enterprise system requirements Web site.

The WebSphere Adapters are not visible when you run ITLM

Procedure

- **Scenario:** The adapter is not visible when you run the IBM Tivoli License Manager (ITLM).
- **Explanation:** If you want to use ITLM with the WebSphere Adapters, you must activate the ITLM file for each adapter.
- **Solution:** Follow the instructions in "Activating IBM License Metric Tool for WebSphere Adapters" on page 1108.

A message flow with an SAPRequest, SiebelRequest, or PeopleSoftRequest node has deployed successfully, but message BIP3540 is issued indicating that connection failed

Procedure

- **Scenario:** From an SAPRequest, SiebelRequest, or PeopleSoftRequest node, an exception is thrown to indicate that the node is unable to make a connection even though the message flow has deployed successfully. The exception contains message BIP3540 with inserted text that indicates that connection failed.
  For example, for SAP, the inserted text is:

```
Exception in connecting to SAP:Connect to SAP gateway failed
Connect_PM GWHOST= invalidhost.test.co, GWSERV=sapgw00, ASHOST= invalidhost.test.co,
  SYSNR=00
LOCATION CPIC (TCP/IP) on local host ERROR partner not reached (host
  invalidhost.test.co, service 3300) TIME Mon Dec 01 16:43:52 2008 RELEASE 640
COMPONENT NI (network interface) VERSION 37 RC -10 MODULE nixxi_r.cpp LINE 8719
DETAIL NiPConnect2 SYSTEM CALL SiPeekPendConn ERRNO 10061 ERRNO TE
```

  For PeopleSoft, the inserted text is:

```
001DOWNbea.jolt.ServiceException: Invalid Session
```

- **Explanation:** The connection details are not verified when the message flow is deployed. For request nodes, the connection is made at first use.
- **Solution:** If you have configured a policy for this adapter, review the connection properties on that policy and review the text in the BIP3540 message to determine if the connection properties are incorrect.

  If no policy exists for this adapter, review the connection properties on the adapter. If the properties are incorrect, correct them and redeploy the adapter. Alternatively, create a new policy with the correct properties to override the properties that are set on the adapter.

You have deployed an SAP inbound adapter but do not receive expected messages

Procedure

- **Scenario:** You have deployed an SAP inbound adapter but do not receive the IDoc messages that you expected to receive.
- **Explanation:** If you have not received IDoc messages from SAP, it is possible that deployment was unsuccessful or the SAP server has not started.
- **Solution:** Check user trace for message BIP3484 occurring at the time of deployment. The adapter component writes diagnostic information to this message, in an insert that begins "CWYAP...". If this message is issued, it explains the cause of the problem.

You have imported an existing project into your workspace, but messages are issued when you try to build SAP message sets

Procedure

- **Scenario:** You have imported an existing project into your workspace, but when you try to build an SAP message set, you see the message set compile error message BIP0182.
- **Explanation:** This error occurs when you choose the option to "Import existing projects into your workspace" from the Import dialog box. By choosing this option when you import, a link is created from the workspace to the existing projects in an external location and a required file is not available to the workspace. To copy the entire project into your workspace, use the option to import the Project Interchange (PI) file.
- **Solution:** When you import an exiting SAP project into your workspace, click **File** > **Import**, expand the **Other** folder, and click **Project Interchange**.

  For more information, see “Migrating development resources to IBM App Connect Enterprise 12.0” on page 62.

An error is issued when you use the message set that is generated by the Adapter Connection wizard

Procedure

- **Scenario:** You run the Adapter Connection wizard and select an inbound SAP IDoc. You run the wizard again, but this time you select an outbound SAP IDoc. When you use the message set that is generated, the following error is issued:

```
'Selector exception caught from generateEISFunctionname', 'commonj.connector.runtime.SelectorException:
commonj.connector.runtime.SelectorException: For the IDoc type SapYwmspgi01, operation key=YWMSPGIWMS
 not
found using the application-specific information {Create={MsgType=, MsgCode=, MsgFunction=}} verify
 appropriate
combination of MsgType, MsgCode, MsgFunction is set in SapYwmspgi01, application-specific information.
```

  - **Explanation:** If you run the Adapter Connection wizard for an inbound SAP IDoc, then you run the wizard again for an outbound SAP IDoc, the outbound IDoc definition replaces the inbound IDoc definition. Information that is stored in the inbound definition is used to map MsgType, MsgCode, and MsgFunction to a method binding. The outbound definition does not contain these mappings, so processing of the inbound IDoc fails.
  - **Solution:** To avoid this error, ensure that inbound and outbound SAP IDocs have different names if they are stored in the same message set.

When you run the SAP samples on Linux or UNIX, IBM App Connect Enterprise does not connect to the SAP gateway

Procedure

- **Scenario:** When you run the SAP samples on Linux or UNIX, the message flow deploys successfully, but a connection is not established between IBM App Connect Enterprise and the SAP system.

  You might see the following error message:

  ```
  message: Connect to SAP gateway failed
  Connect parameters: TPNAME=SAMPRFC GWHOST=sapdev10 GWSERV=sapgw00
  ERROR service 'sapgw00' unknown
  ```

- **Explanation:** For the SAP Java Connector (SAP JCo) to communicate across the network, you need to configure the TCP/IP service. If you have installed a working SAP GUI on your workstation, the TCP/IP service is configured as part of the installation. If you have not installed an SAP GUI, you must configure the TCP/IP service manually so that the SAP samples run successfully.
- **Solution:** Locate the services file in `/etc/services` and edit it so that it includes the appropriate gateway service IDs and the port numbers for the TCP/IP service in the format `sapgw`*`SID`* *`portnumber`*`/tcp`, where *SID* is the SAP system ID.

  For example:

  ```
  sapgw00  3300/tcp
  sapgw01  3301/tcp
  sapgw02  3302/tcp
  ```

  and so on.

  For more information about TCP/IP configuration, see the Network Integration section of the SAP Support Portal.

SAP inbound messages (ALE and BAPI) appear to be missing

Procedure

- **Scenario:** SAP inbound messages (ALE and BAPI) appear to be missing. You might find that every other message does not reach the integration node but no errors are issued.
- **Explanation:** This problem is typically caused when two integration nodes share the same program ID (SAP RFC Destination). For example, a developer has deployed a message flow, but someone else has used the same BAR file without having changed the program ID.
- **Solution:** Ensure that no other integration nodes are running with the same program ID. Use SAP transaction SMGW to determine whether other integration nodes are connected to the SAP system.

When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs

Procedure

- **Scenario:** When two ALE inbound modules use the same RFC program ID with SAP JCo version 3.0.2, NullPointer exceptions are logged in the JCo trace and the Adapter does not receive IDocs.

  The following example shows a typical exception in the JCo trace.

  ```
  JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] Dispatcher.getNextListener() returns
   dispatch next call rfc handle(1) for null
  JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] caught Throwable in DispatcherWorker.run()
   while trying to dispatch a request java.lang.NullPointerException
   at java.util.Hashtable.get(Hashtable.java:518)
   at com.sap.conn.jco.rt.DefaultServerManager
  $DispatcherWorker.run(DefaultServerManager.java:268)
  ```

```
  at java.lang.Thread.run(Thread.java:735)

JCoDispatcherWorkerThread [16:44:42:140]: [JCoApi] Dispatcher.getNextListener() returns no
 calls
```

- **Explanation:** SAP JCo version 3.0.2 does not support the use of two ALE inbound modules with the same RFC program ID.
- **Solution:** To solve this problem, download a hotfix from SAP; the SAP ticket reference number is 584247/2009.

You are using a SiebelInput node with the delivery type set to unordered, and error message BIP3450 is issued with a NullPointer exception

Procedure

- **Scenario:** You are using a SiebelInput node, you have set the delivery type to unordered in the Adapter Connection wizard, and the minimum number of connections is 1 or less. The following exception is shown in user trace: `RecoverableException BIP3450E: An adapter error occurred during the processing of a message. The adapter error message is java.lang.NullPointerException`.
- **Explanation:** When using unordered events, the minimum connections (`MinimumConnections`) and maximum connections (`MaximumConnections`) properties must be greater than 1 for event delivery to be successful.
- **Solution:** Set the `MinimumConnections` and `MaximumConnections` properties on the Adapter Connection wizard to values greater than 1. For example, set the minimum number of connections to 2 and the maximum number of connections to 4.

Error message BIP3450 is issued when you include a JDEdwardsRequest node in a message flow and try to connect to a JD Edwards EnterpriseOne server

Procedure

- **Scenario:** You have created a message flow that contains a JDEdwardsRequest node, but error message BIP3450 is issued, indicating that the node is unable to connect to the JD Edwards EnterpriseOne server.
- **Explanation:** This error message indicates that the JDEdwardsRequest node is trying to connect to the JD Edwards EnterpriseOne server but is unable to do so. This error can be caused by the following situations.
  - The JD Edwards EnterpriseOne server is not running.
  - The JD Edwards adapter has been configured with incorrect connection details; for example, the name of the JD Edwards EnterpriseOne Environment to which to connect.
  - The JDBC drivers that are required to connect to the JD Edwards EnterpriseOne server are missing from the class path. The following table lists the required JDBC driver files for each database.

Database	JDBC driver files	Implementation class
Oracle	tsnames.ora classes12.zip	oracle.jdbc.driver.OracleDriver
SQLServer	sqljdbc.jar	com.ibm.microsoft.sqlserver.jdbc.SQLServerD
AS/400	jt400.jar	com.ibm.as400.access.AS400JDBCDriver
DB2 Type-2 (JDK 1.4/1.5)	db2java.zip	com.ibm.db2.jdbc.app.DB2Driver

Database	JDBC driver files	Implementation class
DB2 Type-4 (JDK 1.4/1.5)	db2jcc.jar db2jcc_license_cu.jar	com.ibm.db2.jcc.DB2Driver
DB2 Type-4 (JDK 1.6)	db2jcc4.jar	com.ibm.db2.jcc.DB2Driver

- **Solution:** Ensure that the following conditions have been met.
  - The JD Edwards EnterpriseOne server is running.
  - The JD Edwards adapter has been configured with the correct connection details.
  - The drivers that are required to connect to the JD Edwards EnterpriseOne server are in the class path.

Resolving other problems when developing message flows

Use the advice given here to deal with problems that can arise when developing message flows, and that are not covered in the specific categories listed in "Resolving problems when developing message flows"

About this task

- "The values of your promoted properties are lost after editing" on page 2951
- "The Message Flow editor experiences problems when opening a message flow, and opens in error mode" on page 2951
- "A message flow has subflows with the same user-defined property set to different values, but only one value is set at run time" on page 2952
- "You want to move resources to a new broker schema that you have created but it is not visible in the Application Development view" on page 2952
- "A selector exception is raised when you use WebSphere Adapters" on page 2952

The values of your promoted properties are lost after editing

Procedure

- **Scenario:** You edited a message flow using the Message Flow editor, and the values of your promoted properties are lost.
- **Explanation:** The values of promoted properties for nodes with more than a single subflow definition (that is, two identically named subflows in the same project reference path) are lost if the flow is edited and saved.
- **Solution:** To avoid this problem, ensure that each subflow in your project has a different name.

The Message Flow editor experiences problems when opening a message flow, and opens in error mode

Procedure

- **Scenario:** You attempt to open an existing message flow in the Message Flow editor and it opens in read-only error mode, displaying a list of parsing or loading errors. The message flow is not open and a message is displayed indicating that the message flow file is not valid.
- **Explanation:** The message flow file is unreadable or is corrupted, and the Message Flow editor cannot render the model graphically.
- **Solution:** Contact IBM Customer Support for assistance with the corrupted file.

A message flow has subflows with the same user-defined property set to different values, but only one value is set at run time

Procedure

- **Scenario:** You have a message flow that contains identical subflows. Each subflow has the same user-defined property (UDP), but with different values. At run time, only one of the values is set.
- **Explanation:** A UDP has global scope and is not specific to a particular subflow. If you reuse a subflow in a message flow, and those subflows have identical UDPs, you cannot set the UDPs to different values.
- **Solution:** If you need to set a different value for each subflow, use a different UDP for each subflow.

You want to move resources to a new broker schema that you have created but it is not visible in the Application Development view

Procedure

- **Scenario:** You have created a new broker schema and you want to move a resource to it, but the schema is not visible in the Application Development view.
- **Explanation:** If category mode is selected, you cannot see the broker schema in the Application Development view.
- **Solution:** To move resources to a broker schema that you have created, take one of the following steps.
  - Click **Hide Categories** () on the Application Development view toolbar. The new broker schema appears in the Application Development view and you can drag resources onto it.
  - Right-click a resource, click **Move**, then select the schema that you have created. When you click **OK**, the resource is moved to the selected schema.

A selector exception is raised when you use WebSphere Adapters

Procedure

- **Scenario:** You run the Adapter Connection wizard for an inbound IDOC, then you run the wizard again for an outbound IDOC. When the message set is generated, you see the following error message:

```
'Selector exception caught from generateEISFunctionname' ,
'commonj.connector.runtime.SelectorException:
commonj.connector.runtime.SelectorException: For the IDoc type
SapYwmspgi01, operation key=YWMSPGIWMS not found using the
application-specific information {Create={MsgType=, MsgCode=, MsgFunction=}}
verify apropriate combination of MsgType,MsgCode, MsgFunction is set in
SapYwmspgi01, application-specific information.
--
```

- **Explanation:** When you run the Adapter Connection wizard for an inbound IDOC, then you run the wizard again for an outbound IDOC, the outbound IDOC definition replaces the inbound IDOC definition. Information that is stored in the inbound definition is used to map MsgType, MsgCode, and MsgFunction to a method binding. The outbound definition does not contain these mappings, so processing of the inbound IDOC fails.
- **Solution:** To avoid this error, ensure that inbound and outbound IDOCs have different names if they are stored in the same message set.

Resolving problems when deploying message flows or message sets

Use the advice given here to help you to resolve common problems that can arise when you deploy message flows or message sets.

Initial checks

Procedure

1. To debug problems when deploying, check the following logs:
   - The administration log
   - The local error log (the Windows Event log or the syslog)
   - The IBM MQ logs

   These logs might be on separate computers, and must be used with the IBM App Connect Enterprise Toolkit output to ensure that the deployment was successful.

   Use the **mqsilist** command to check that the deployment was successful, or look in the Windows Event or Administration Log view.
2. Use this checklist when you have deployment problems:
   - Make sure that the mode that your integration node is working in is appropriate for your requirements. See “Operation modes” on page 3.
   - Make sure that the client connection to a remote queue manager is running.
   - Make sure that channels are running.
   - Display the channel status to see if the number of system messages sent increases.
   - Check the channel from the remote end.
   - Check the queue manager name.
   - Determine whether the channel is a cluster channel.

Common problems

About this task

- "You receive exception ('MQCC_FAILED') reason '2042' ('MQRC_OBJECT_IN_USE')" on page 2961

Resolving problems that occur when preparing to deploy message flows

Use the advice given here to help you to resolve common problems that can occur during preparations to deploy message flows or message sets.

An error is issued when you add a message set to a BAR file

Procedure

- **Scenario:** An error is issued when you add a message set to a BAR file.
- **Explanation:** After you create a BAR file and add a message set project to it, two files are created in the BAR file: `messageset.user.txt` and `messageset.service.txt`. The `user.txt` file contains user log information, such as warning message BIP0177W, which states that the dictionary that you have created is not compatible with earlier versions.
- **Solution:** Use the information in the `user.txt` file to diagnose the error. The `service.txt` file contains detailed information that is used by the integration node, and can be used by the IBM Support Center to diagnose problems.

You cannot drag a BAR file to an integration node

Procedure

- **Scenario:** You cannot drag a BAR file to an integration node.
- **Explanation:** BAR files can be deployed only on an integration server.
- **Solution:** Select an integration server in the deploy dialog.

You cannot deploy a message flow that uses a user-defined message flow

Procedure

- **Scenario:** You have created a message flow that contains an input node in a user-defined node project. However, you cannot deploy a message flow that uses this node.
- **Explanation:** Validation, compilation, and deployment do not recognize that a user-defined message flow contains an input node.
- **Solution:** To work around the problem, add a dummy input node to the flow that you intend to deploy.

The compiled message flow file (`.cmf`) has not been generated

Procedure

- **Scenario:** The compiled message flow file (`.cmf`) has not been generated.

  Therefore, it is not added to the BAR file, and cannot be deployed.
- **Explanation:** When you create files that define message flow resources in the IBM App Connect Enterprise Toolkit, the overall file path length of those files must not exceed 256 characters, because of a Windows file system limitation.

  If you have a message flow that includes resource files that have a path length that exceeds 256 characters, the message flow cannot be compiled when you try to add it to a BAR file, and therefore cannot be deployed.

  You might also be affected by this restriction when you use the Adapters Connection wizard; names discovered and returned by the EIS can be very long. The Adapters Connection wizard attempts to write these files to the message set on the local file system, but their path exceeds the operating system limit, and the files appear corrupted.

- **Solution:** To ensure that the path length does not exceed 256 characters, use names that are as short as possible for all resources; for example:
  - The installation path
  - Project names and broker schema names
  - ESQL and mapping file names
  - Inbound and outbound adapter files

You cannot deploy a message flow that contains restricted resources

Procedure

- **Scenario:** An error is issued when you deploy a BAR file containing resources that are not valid for your runtime operation mode.
- **Explanation:** In express mode and scale mode, the integration node operates with a limited set of nodes. If a BAR file contains nodes that are restricted in the runtime operation mode, you cannot deploy the BAR file.
- **Solution:** Rework your message flow to use resources that are valid for your runtime operation mode; see "Restrictions that apply in each operation mode" on page 5. Alternatively, contact your IBM representative to upgrade your license.

Resolving problems that occur during deployment of message flows

Use the advice given here to help you to resolve problems that can arise during deployment of message flows or message sets.

Procedure

You receive a warning message about the mode of your integration node

About this task

Message BIP1806

- **Scenario:** Warning message BIP1806 is displayed.
- **Explanation:** Your integration node has not been started.
- **Solution:** Check that the integration node is started: if it is not started, start it by using the **`mqsistart`** command.

Message BIP1821

- **Scenario:** Warning message BIP1821 is displayed.

- **Explanation:** Your integration node is running in a mode that restricts the number of integration servers that you can use; see "Restrictions that apply in each operation mode" on page 5.
- **Solution:** Contact your IBM representative to upgrade your license, or remove the required number of integration servers; see "Deleting an integration server" on page 175.

The message flow deploys on the test system, but not elsewhere

Procedure

- **Scenario:** The message flow that you have developed deploys on the test system, but not elsewhere.
- **Solution:** Carry out the following checks:
  - Ensure that you have verified the installation on the target system by creating and starting an integration node, and deploying a single integration server. These actions confirm that the integration node is correctly defined.
  - Ensure that the BAR file's `broker.xml` file contains references to the correct resources for the new system.
  - Ensure that any referenced message sets are deployed.
  - If database resources or user-defined nodes are not accessible or authorized from the target system, the deploy fails. On distributed systems, ensure that you have defined either ODBC or JDBC connections to your databases so that they can be accessed from the integration node. Also, set the integration node environment to allow access to the databases. On Linux or UNIX systems, you might have to run a database profile.
  - Any user-defined extensions that you are using in your message flow might not load if they cannot be found, or are not linked correctly. Consult the documentation for your operating system for details of tools that can help you to check the binary files of your user-defined extension.

Error messages about your integration node mode are issued when you create an integration server

About this task

Message BIP1825

- **Scenario:** Error message BIP1825 is displayed.
- **Explanation:** The integration server cannot be created because the maximum number of integration servers for the mode of the target integration node has been reached, and creating the integration server causes this limit to be exceeded; see "Restrictions that apply in each operation mode" on page 5. The integration server was not created.
- **Solution:** Reuse an existing integration server, or delete an existing integration server and try the command again; see "Deleting an integration server" on page 175. Alternatively, contact your IBM representative to upgrade your license.

Error messages about your integration node mode are issued when you deploy

About this task

Message BIP1829

- **Scenario:** Error message BIP1829 is displayed.
- **Explanation:** The BAR file cannot be deployed because the maximum number of integration servers for the mode of the target integration node has been reached; see "Restrictions that apply in each operation mode" on page 5. The BAR file was not deployed.
- **Solution:** Delete an existing integration server and try the command again; see "Deleting an integration server" on page 175. Alternatively, contact your IBM representative to upgrade your license.

You create a policy, then deploy a message flow and inbound adapter, but the deployment fails

Procedure

- **Scenario:** You create a policy, then deploy a message flow and inbound adapter, but the deployment fails.
- **Explanation:** When you create a policy for an adapter that has not yet been deployed, the connection properties are not fully validated until you deploy the adapter and the message flow that uses that adapter. Therefore, the properties that you set on the policy might be invalid.
- **Solution:** Inspect the error message that is returned from the deployment and determine whether the deployment failed because of invalid connection properties on the policy. If so, update the values in the policy, then redeploy the policy project.

You have created a WebSphere Adapters message flow that uses secondary adapters, and a naming clash has occurred in the secondary adapters or message sets

Procedure

- **Scenario:** You have created a WebSphere Adapters message flow that uses secondary adapters, and a naming clash has occurred. You need to know which adapters and message sets the WebSphere Adapters node is using, and if the method names in the adapters or the message type names in the message sets are clashing.
- **Explanation:** Method names must be unique across all primary and secondary adapters, and the message set that is created must not contain any types that share the same name and namespace of existing message sets. Information about secondary adapters and message sets is written to user trace at the following stages.
  - When the node is first deployed, information about the current set of secondary adapters and message sets is reported.
  - When adapters and message sets are deployed subsequently, information about them is reported at the time of deployment.
  - When the message flow, integration node, or integration server is stopped then restarted, information about the entire set of secondary adapters and message sets is written to user trace, including those secondary adapters and message sets that are deployed after the message flow was first deployed.

  This information is reported by the following messages.

  - BIP3432 and BIP3434 list the adapters and message sets that are available when the node is deployed or when the integration node, integration server, or message flow are restarted.
  - BIP3433 reports that a secondary adapter is being deployed and added to the set.
  - BIP3435 reports that a secondary message set is being deployed.
  - BIP3436 identifies the message set in which each type is being defined.
  - BIP3437 identifies message sets that attempt to redefine a type that is already defined.
  - BIP3438 identifies the adapter in which each method is being defined.
  - BIP3439 identifies adapters that attempt to redefine a method that is already defined.

  Messages BIP3436, BIP3437, BIP3438, and BIP3439 are issued when a message is processed to report whether definitions are used for that message.

- **Solution:** The following steps describe how to use user trace when working with secondary adapters.
  - **When you deploy the flow for the first time**
    1. Start user trace.
    2. Deploy your message flow, primary adapter, primary message set, and any secondary adapters or message sets that are ready.
    3. Read user trace, looking out for the messages described.
    4. Optional: Reset user trace, stop and restart the message flow, then read user trace again, looking out for the messages described.
  - **When you add new adapters and message sets**
    1. Start user trace.
    2. Deploy the secondary adapters and message sets.
    3. Read user trace, looking out for the messages described.
    4. Optional: Reset user trace, stop and restart the message flow, then read user trace again, looking out for the messages described.

  When you have identified where the naming clash has occurred, you can ensure that names are unique by editing them in the following ways:

  - Edit method names by clicking **Edit Operations** on the **Service Generation and Deployment Configuration** panel of the **Adapter Connection** wizard.
  - Edit the namespaces of the types in the message set by using the Business Object Namespace control on the **Adapter Connection** wizard.

Deployment fails when you have circular project dependencies

Procedure

- **Scenario:** Circular project dependencies exist in the projects that you are deploying and deployment fails.
- **Explanation:** A circular project dependency occurs when two or more projects depend on each other.

  For example, File A in Project X depends on File B in Project Y, and File C in Project Y depends on File D in Project X. For Project X to build successfully, Project Y must be built first so that Project X can resolve the dependency on File B. However, for Project Y to build successfully, Project X must be built first so that Project Y can resolve the dependency on File D. The IBM App Connect Enterprise Toolkit is based on the Eclipse platform, which does not support circular project dependencies.
- **Solution:** To deploy successfully, avoid circular project dependencies.

  For example, if Project X depends on File B in Project Y, and Project Y depends on File D in Project X, move File B to Project Z. The projects must be built in the order Project Z, Project X, then Project Y so that the dependencies can be resolved.

Error messages are issued when you deploy

About this task

Additional error messages that might be generated during a deployment are explained in this section.

Message BIP1106 with IBM MQ reason code 2030

- **Scenario:** Error message BIP1106 is issued with reason code 2030 when you are deploying a large message set.
- **Explanation:** The size of the message exceeds the maximum message length of the transmission queue to the integration node queue manager.

- **Solution:** Increase the maximum message length for the transmission queue using the IBM MQ **alter qlocal** command, where the maximum message length (maxmsgl) is in bytes:

```
alter ql(transmit_queue_name) maxmsgl(104857600)
```

For more information about this command, see the *System Administration Guide* section of the IBM MQ product documentation online.

Message BIP2066

- **Scenario:** You have initiated a deployment request; for example, you have deployed a BAR file to an integration server. Error message BIP2066 is returned one or more times.
- **Explanation:** The deployment request was not acknowledged by the integration server before the sum of the values for the integration node timeout parameters **ConfigurationChangeTimeout** and **InternalConfigurationTimeout** expired.
- **Solution:** Increase these timeout values by specifying the **-g** and **-k** parameters of the **mqsicreatebroker** command (command).

Message BIP2080

- **Scenario:** The integration node has started an integration server; for example, if you have issued **mqsistart** for the integration node, or an error has occurred and the integration server is being recovered. Error message BIP2080 is displayed one or more times.
- **Explanation:** The internal configuration request was not acknowledged by the integration server before the value of the **InternalConfigurationTimeout** (default 60 seconds) expired.
- **Solution:** Change the configuration timeout by specifying the **-k** parameter of the **mqsicreatebroker** command (command).

Message BIP2241

- **Scenario:** Error message BIP2241 is displayed.
- **Explanation:** You are attempting to deploy a message flow containing a node that is not available on the target integration node.
- **Solution:** Ensure that the version of the IBM App Connect Enterprise Toolkit in which the message flow has been developed matches the version of the integration node to which the message flow is being deployed. If the message flow is using a user-defined node, or a node supplied in a SupportPac, ensure that the runtime node implementation has been correctly installed on the computer on which the integration node is running. If your message flow includes a user-defined node, see “Installing user-defined extension runtime files on an integration node” on page 2455. If your message flow includes a node provided in a SupportPac, see the installation information, if supplied, for the SupportPac.

Message BIP2242

- **Scenario:** Error message BIP2242 is displayed.
- **Explanation:** The deployment request (configuration change) was not accepted before the timeout value set by the integration node parameter **ConfigurationChangeTimeout** expired. This configuration timeout value must be long enough for the message flow to complete processing its current message, then accept the deploy request; the default is 300 seconds.
- **Solution:** Set the configuration timeout values by specifying the **-g** and **-k** parameters of the **mqsicreatebroker** command.

Message BIP3226

- **Scenario:** Error message BIP3226 is displayed; for example:

```
(Semipersistent_Compute1.Main, 27.89) : Array index evaluated to '0' but must
evaluate to a positive, nonzero integer value.
```

The first insert in BIP3226 (in this example, `Semipersistent_Compute1.Main`) identifies the node and routine in which the statement occurs. The second insert (in this example, `27.89`) identifies the approximate line and column of the index value shown in the third insert (in this example, '0').

- **Explanation:** The validity of using a field reference index of zero was corrected in WebSphere Message Broker Version 7.0. If you have statements in your ESQL modules that include an index of zero, error BIP3226E is generated.

  For example, your ESQL module might contain the following statement:

```
SET OutputRoot.XMLNSC.Top.A[0].B = 42;
```

- **Solution:**

  You must correct all ESQL statements that use an index of zero to use an index of 1. Statements might use a variable as well as a literal value for the index; check for both possible situations. For example, your changed code might read:

```
SET OutputRoot.XMLNSC.Top.A[1].B = 42;
```

Tagged/Delimited String (TDS) Validator error

- **Scenario:** You try to deploy a message set with a TDS wire format that has an error.
- **Explanation:** The following extract from an error log illustrates what you might see for a TDS Validator error. In this case, the cause of the problem is that the element Town does not have a tag defined.

```
TDS Extractor Trace File
========================

Beginning Extract..

Extracting Identification Info
Extracting Project Info
Extracting Messages
Extracting Elements
Extracting Compound Types
Extracting Type Members
Extracting Type Members
Extracting Type Members
Extracting Type Members
Extracting Type Members
Beginning Indexing..

Creating Member IDs to Tags Index Table.

Beginning Validation..

Validating Project
Validating Types
ERROR: TDSValidator::ValidateTypeMemberSimpleElement:
  Simple elements in a type with Data Element Separation attribute = Tagged
  Delimited must have the following attribute set:
  Element Level - Tag
(Element ID: Town)
(Type ID: AddressType)
Return Code: -80

Validating Messages

Trace Info
===========
EXCEPTION: TDSValidator::Validate:
  TDS Validation failed.
    1 errors
    0 warnings
Return Code: -1
```

- **Solution:** Use the information in the error log to correct the problem.

Resolving problems that occur after deployment of message flows

Use the advice given here to help you to resolve common problems that can arise after deploying message flows or message sets.

You receive exception ('MQCC_FAILED') reason '2042' ('MQRC_OBJECT_IN_USE')

Procedure

- **Scenario:** This exception message occurs when the following conditions are met:
  a) You have a message flow containing a SOAPRequest node that is using the JMS Transport.
  b) The message flow has additional instances defined, and is running under load.
  c) You are using the IBM MQ JMS provider.
  d) You have not specified a reply-to destination, and so are using JMS temporary dynamic queues.
- **Explanation:** If you do not specify a reply-to destination, and you are using the IBM MQ JMS provider in a message flow with additional instances, you must configure JMS temporary dynamic queues before deploying your message flow.
- **Solution:** Complete the steps in the topic Configuring JMS temporary dynamic queues for the JMS provider.

Your XSLTransform node does not work after deployment

About this task

Two scenarios explain why your XSLTransform node might not work after deployment:

Error messages are displayed indicating that the style sheets were not found

Procedure

- **Scenario:** Error messages are displayed indicating that the style sheets were not found.
- **Explanation:** This error message is displayed if the integration node cannot find the style sheets or XML files required, or if the content of a style sheet or XML file is damaged and therefore no longer usable. This error message can happen if a file system failure occurs during a deployment.
- **Solution:** If the style sheets or XML files are damaged, redeploy the damaged style sheets or XML files.

What to do next

You get unexpected transformation results

- **Scenario:** You get unexpected transformation results.
- **Explanation:** For complex message flows, incompatibility might arise among style sheets and XML files after a deployment. The two typical reasons for this error are:
  - Only part of the cooperating style sheets or XML files are deployed and updated (a file system failure could cause this failure).
  - Multiple XSLTransform nodes that are running inside the same integration server are supposed to use compatible style sheets, but are using different versions to process the same incoming message.
- **Solution:** If only part of the cooperating style sheets or XML files are deployed and updated, resolve all incompatibility by redeploying the compatible versions. To avoid multiple XSLTransform nodes using different versions of the style sheet, pause relevant message flows in the target integration server before performing the deployment, then restart the flows.

Resolving problems that occur when debugging message flows

Advice on dealing with some of the common problems that you see when debugging message flows.

About this task

Resolving problems that occur when starting and stopping the debugger

Use the advice given here to help you to resolve common problems that can arise when you debug message flows.

An endless "waiting for communication" progress bar is displayed when you start the debugger

Procedure

- **Scenario:** After you click **Start Debugging**, you get an endlessly cycling progress bar entitled "waiting for communication". The "debug session started" message is not displayed in the information pane.
- **Explanation:** If the message flow has nodes with ESQL statements, the flow might not deploy even if the statements are correct syntactically.
  This situation can occur, for example, because of multiple declarations or uninitialized variables (that is, semantic problems that the syntax parser does not pick up).

  Always check the IBM App Connect Enterprise Toolkit Administration log to confirm that the debugged version of your message flow has deployed successfully; it has the same name as the original message flow, with the suffix _debug_.

  If the message flow does not deploy properly, the debugger cannot establish communication with the flow, and you see the endless progress bar.
- **Solution:** Click **Cancel** to clean up and return to a good state, then fix your errors and try again.

  As a check, see if your flow can deploy without the debugger.

The debugger seems to stop

Procedure

- **Scenario:** You are debugging a message flow and continue after encountering a breakpoint. However, nothing seems to happen and after about a minute, a progress bar appears, indicating that the debugger is waiting for communication.
- **Explanation:** This situation can occur for the following reasons:
  - The message flow might have encountered a time-intensive operation, such as a huge database query, and you must wait for the flow to complete the action.
  - The integration node ended, or some other extraordinary condition occurred, and communication was lost. In this case, click **Cancel** to stop the debug session.

The session ends abnormally while debugging

Procedure

- **Scenario:** After debugging a message flow, the session ends abnormally and you still have the debug instance of the message flow (mf_debug_) deployed to the integration node's integration server. You are concerned that this is going to affect the operation of the flow, and want to put the integration server back to its original state.
- **Explanation:** The orphaned message flow should behave as the flow would have done normally, and the Debug nodes have no effect on message processing.

  If you have a small number of nodes in the message flow, corrective action makes no noticeable difference to the flow, apart from its name. However, if you have a large message flow (that is, more than 15 nodes or several subflows), take the corrective action described later in this section, because the performance of message processing might be affected.
- **Solution:** Redeploy the integration node.

  A full redeploy of the integration node should replace the orphaned flow with the original message flow. If this action has no effect, remove the orphaned flow from the integration server, deploy, then add the flow and deploy to restore the original state of the integration node as it was before the debugging session.

An error message is displayed indicating that the debug session cannot be launched

Procedure

- **Scenario:** You try to relaunch or invoke a new debug session but when you click the green **Debug** icon  , an error message is displayed stating: `Cannot launch this debug session`.
- **Explanation:** When you click **Debug** , it relaunches the last debug session. It fails if you have not created a debug session previously. It also fails if the integration node and integration server that were attached previously in a debug session are no longer running, or have been restarted; the session cannot be reattached without re-selection of the integration node and integration server process instance.
- **Solution:**
  a) Close the error message and click the arrow immediately to the right of the **Debug** icon.
  b) Re-select or modify the integration node and integration server information from the previous debug launch configuration by clicking **Debug** in the menu and selecting the previous debug launch configuration. See “Attaching the flow debugger to an integration server for debugging” on page 665 for more information.

Errors are generated when you copy a message map into an application, library, or integration project

Procedure

- **Scenario:** You are copying a message map into an application, library, or integration project and errors have appeared in the task list.
- **Explanation:** The application, library, or integration project did not have the correct references set before you copied the message mapping.
- **Solution:** These errors remain in the task list, even if you reset the project references immediately after copying. Therefore, you must perform a clean build of the application, library, or integration project.

Resolving problems when debugging message flows

This topic contains advice for dealing with some of the common problems that can arise when debugging message flows.

The debugger does not pause at the next breakpoint

Procedure

- **Scenario:** The message flow debugger does not pause at the next breakpoint in your message flow.
- **Solution:** Perform the following checks:
  a) Check whether your DataFlowEngine is running; if it is not, restart it.
  b) Check the input queue. If your input queue has the leftover messages from the previous time that you used the debugger, clear them before you send a new message.

The message does not stop executing at any breakpoint

Procedure

- **Scenario:** The message does not stop executing at any breakpoint after you attach to the debugger.
- **Explanation:** This error could be caused by a timing problem, or if you have set the wrong parameters for the debug session.
- **Solution:** Perform the following steps.
  a) Check your launch configuration settings, ensuring that you have specified the correct Flow Project, HostName and Flow Engine for the debug session.
  b) Restart the debug session.

Editing problems occur in the Message Flow editor

Procedure

- **Scenario:** While debugging a message flow, editing problems occur when you are using the Message Flow editor.
- **Solution:** Do not attempt to edit the message while the flow debugger is attached. To edit a message flow, detach the debugger, edit the message flow, then redeploy the message flow.

Editing the MQ message descriptor (MQMD) causes unexpected behavior in the debugger

Procedure

- **Scenario:** You edit properties of the message MQMD descriptor in the Message Set editor, but this causes unexpected behavior in the debugger.
- **Explanation:** If you edit the content of the MQMD descriptor, these fields take a certain range of values.

  You need to know these ranges before editing the properties. Unless you explicitly specify the value of these fields, they take default values, and certain fields might not have been specified in the message. The values in the fields that are not set explicitly in the message are default values; do not change these unless you are aware of their importance or the possible range of values.

You cannot see the message content when debugging your message flow

Procedure

- **Scenario:** You are using the message flow debugger, and you can see the message passing through the message flow, but you cannot see the content of the message.
- **Solution:** Open the Flow Debug Message view by clicking **Window** > **Show View** > **Other** > **Message Flow** > **Flow Debug Message**, then **OK**.

An exclamation mark appears above a node during debugging

Procedure

- **Scenario:** In the Message Flow editor, an exclamation mark (!) is displayed above a node during debugging.
- **Explanation:** An exception has occurred in the node during debugging.
- **Solution:** Look under the ExceptionList in the **Variables** view of the Debug perspective to find out what error has occurred.

The message does not stop processing at breakpoints

Procedure

- **Scenario:** Message processing continues when a breakpoint is encountered.
- **Explanation:** This error could be caused by a timing problem, or if you have set the wrong parameters for the debug session.
- **Solution:** Check your launch configuration setting. Ensure you have specified the correct Flow Project, HostName and Flow Engine for the debug session. Restart the debug session.

You cannot see where the debugger is in the Graphical Data Mapping editor

Procedure

- **Scenario:** The Graphical Data Mapping editor has opened in the IBM App Connect Enterprise Toolkit, but it is unclear where the debugger is in the map.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

When debugging a message map, the debugger does not move to the next field

Procedure

- **Scenario:** You are debugging a message map, and the debugger does not move to the next field. You have to click the **Step over** button multiple times.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

When debugging a message map, the debugger does not move out of the mapping node

Procedure

- **Scenario:** You are debugging a message map, and the debugger does not move out of the message map.
- **Explanation:** The source lookup path for the message map file is not configured correctly.
- **Solution:** Check your debug launch configuration settings and ensure you have configured the source lookup path for the message map file correctly.

The message flow stops at a collector node

Procedure

- **Scenario:** Message processing stops after selecting the Step into Source Code icon on a Collector node.
- **Explanation:** The collector node is a multithreaded node and the thread is ended by selecting Step into Source Code.
- **Solution:** Set a breakpoint manually after the collector node.

Resolving problems that occur after debugging:

This topic contains advice for dealing with some of the common problems that can arise after debugging message flows.

You cannot change a message flow after debugging

Procedure

- **Scenario:** You were debugging, but now your message flow seems to be stuck. When you put a new message in, nothing happens.
- **Explanation:** This might be because a message was backed out, but you have not set the **`Backout requeue name`** property of your input queue.
- **Solution:** Set the **`Backout requeue name`** property to a valid queue name (such as the name of the input queue itself) and your flow will become unstuck.

You redeployed a debugged message flow but deployment hangs

Procedure

- **Scenario:** You found problems in your message flow by using the debugger. You changed your message flow and then redeployed it, but now the deployment hangs.
- **Solution:** Make sure that when you redeploy the flow to an integration server, the integration server is not still attached to the debugger.

Resolving diagnostic data collection errors

About this task

Procedure

- **Scenario:** When you collect data by using the **aceDataCollector** command, you receive the following error:

```
Access is denied.
The directory or file cannot be created.
The directory or file cannot be created.
The directory or file cannot be created.
The directory or file cannot be created.
The directory or file cannot be created.
The directory or file cannot be created.
 - Capturing environment and version data

The system cannot find the path specified.
The system cannot find the path specified.
```

- **Explanation:** The user who is running the **aceDataCollector** command does not have either read or write access to the current working directory.
- **Solution:** Ensure that the user who is running the **aceDataCollector** command had the necessary permissions for the directory from which the command is run.

Resolving problems when developing message models

This topic contains advice for dealing with some common problems that can arise when working with message sets.

About this task

Message definition files:

Other:

Your message definition file does not open

Procedure

- **Scenario:** You have created a complex type with a base type that causes the types to be recursive, and now your message definition file does not open.
- **Solution:** Recover the last saved version of your message definition file from the local history or from your repository.

A message definition file error is written to the task list

Procedure

- **Scenario:** An error is written to the task list indicating that the message definition file has been corrupted.
- **Explanation:** This error message appears when the base type of the complex type is itself, or a circular definition of two or more simple types. This type of recursion is not supported.
- **Solution:** Examine your code and ensure that the type of recursion described above does not occur.

Your physical format property values have reverted to defaults

Procedure

- **Scenario:** Someone has sent you a message definition file. You have added it to a message set project, but all the physical format property values have reverted to defaults.
- **Explanation:** You cannot transfer message definition files on their own in this way because these files are not entirely independent.

  It is the `messageSet.mset` file that lists all the physical formats that are applicable to a message set.

  For example, if you have a message set *A* with CWF format *Binary1*, and a message set *B* without any physical formats, a message definition file copied from *A* to *B* does not show *Binary1* as a known physical format (although the properties are still in the message definition file).
- **Solution:** If possible, request the entire message set project.

  Alternatively, add physical formats to the receiving message set so that they match the originating message set. Then any dormant properties become visible. However, make sure that the message set level physical format properties match, or those object properties that inherit defaults from the message set level will be incorrect.

You are unable to enter text in the Message Definition editors

Procedure

- **Scenario:** You are unable to enter text in the Message Definition Overview editor, or change the text of an enumeration or pattern facet in the Message Definition Properties editor.

- **Solution:** The Message Definition editors are cell editors, and to enable them you have to double-click the cell.

  The first click selects the cell; the second click puts the cell into active state, allowing you to edit it.

Objects in your message definition file are not listed in alphabetic order

Procedure

- **Scenario:** The objects in your message definition file (`.mxsd`) are listed in the order that you created them, but you want them to be in alphabetic order.
- **Solution:**

  a) In the Integration Development perspective, double-click the message definition file to open it in the Outline view.

  b) Click **az** (at the top of the Outline view).

  This action changes the order of objects within each of the message definition file's folders (the Messages, Types, Groups, Elements, and Attributes folders) to be alphabetic.

Error messages are written to the task list after you have imported related XML Schema files

Procedure

- **Scenario:** When you have finished importing a collection of related XML Schema files, you find that the IBM App Connect Enterprise Toolkit task list contains error messages for the message set project, indicating that referenced types or other objects cannot be found.
- **Explanation:** These errors are typically an indication that one message definition file includes or imports another message definition file, but that the Message Definition editor is unable to resolve the specified link.
- **Solution:**

  – Check that a message definition file exists in the IBM App Connect Enterprise Toolkit for each of the related XML Schema files that you are importing. If the new message definition files have been created, they appear in the Application Development view.

  – Using the Properties Hierarchy in the Message Definition editor, check that any message definition files that include or import other message definition files are correctly locating the target files.

  One common scenario is where you have two XML Schema files *X* and *Y*, which both exist in the same directory in the file system but where *X*, which includes *Y*, is defined with a real target namespace, while *Y* is defined in the notarget namespace. After you import the two files, *X* is placed into the location that is determined by its namespace, but *Y* is placed into the default namespace location used for files defined in the notarget namespace. This situation causes the link between the two files to break, and you must modify *X* so that it correctly includes *Y* in its new relative location.

A group contains two different elements with the same XML name in the same namespace

Procedure

- **Scenario:** A warning is written to the task list because a group contains two different elements that have the same XML element name, in the same namespace.

- **Explanation:** When a message that has an XML physical format is validated, the validation includes a test to identify any part of a message definition where the parser might not be able to uniquely determine which element is represented by the XML name.

  This test generates a warning when a group contains two different XML elements with the same element name in the same namespace, even in cases where the duplication is legitimate.
- **Solution:** Determine whether the duplication that is identified in the warning message is in fact a problem that needs to be corrected, or whether it has arisen because of two elements on opposite sides of a choice sharing the same XML name, in which case the duplication is legitimate because no ambiguity can occur.

You are unable to set up a message definition file to include a message definition file within another message definition file

Procedure

- **Scenario:** You are unable to set up the include property of a message definition file to include a second message definition file that is included within another message definition file.
- **Explanation:** A message definition file can reference objects in another message definition file only if the first file references the second file directly.
  For example, if three message definition files exist, *A*, *B* and *C*, where *A* includes *B* and *B* includes *C*, *A* can reference objects in *B*, and *B* can reference objects in *C*, but *A* cannot reference objects in *C*.

  You might also encounter this problem after importing XML Schema files that have nested includes.
- **Solution:** You can work around this problem by including the message definition file directly, which, in the above example, would mean including *C* directly in *A*.

  Alternatively, you can define all the message set definitions that are in *C* directly in *B*, then delete *C* so that *A* needs to include only *B*.

You want to make the Properties tab the default tab in the Message Definition editor

Procedure

- **Scenario**: When using the Message Definition editor to edit your message definition files, you prefer to use the **Properties** tab rather than the **Overview** tab, but the **Overview** tab is always selected as the default.
- **Solution:** Use the IBM App Connect Enterprise Toolkit preferences to choose the **Properties** tab as the default tab:
  a) From the menu, click **Windows** > **Preferences**.
  b) Expand Integration Development, then **Message Set**.
  c) Expand **Editors** and select **Tab Extensions**.
  d) In the **Message Definition Editor** section, select the **Properties** and the **Overview** check boxes.
  e) Click **OK**.

Error message BIP5410 is issued because a union type cannot be resolved

Procedure

- **Scenario:** Error message BIP5410 is raised indicating that an element or attribute is based on a union type and that the element or attribute could not be cast to any member of the union.

- **Explanation:** When parsing an element or attribute that is based on a union type, the MRM XML parser uses an xsi:type attribute, where present, to resolve the union.

  If an xsi:type attribute is not present, or an attribute is being parsed, the parser tries each union member type in turn, attempting to cast to the associated simple type, until the cast succeeds. The order of precedence for the attempted cast is the order in which the member types are listed in the message model, under the union type, in the Outline view.

  If the data cannot be cast to any of the simple types within the union, the union cannot be resolved and a parser error is reported.
- **Solution:** Perform the following checks:
  - Check that the message contains a valid value for the element or attribute that is identified in the error message.
  - Check that the member types of the union that are identified in the error message contain the correct list of simple types.
  - If possible, use an xsi:type attribute to resolve the union explicitly.

Error message BIP5395 is issued because an xsi:type attribute value does not correspond to a valid member type of the union

Procedure

- **Scenario:** Error message BIP5395 is issued, indicating that an element is based on a union type that has an xsi:type attribute with a value that should resolve the union explicitly to one of its modeled member types and that the xsi:type attribute value does not correspond to a valid member type of the union.
- **Explanation:** When parsing or writing an element or attribute that is based on a union type, the MRM XML parser or writer uses an xsi:type attribute, where present, to resolve the union.

  The parser resolves the value of the xsi:type attribute to a simple type in the dictionary and checks that the simple type is a valid member type of the union.

  If the xsi:type attribute identifies a type that is not a member type of the union, it reports an error.
- **Solution:** Perform one of the following steps:
  - Modify the message so that the xsi:type attribute identifies a valid member type of the union.
  - Check that the member types of the union identified in the error message contain the correct list of simple types.

Error message BIP5396 is issued because a data type does not correspond to any of the valid data types of the union

Procedure

- **Scenario:** Error message BIP5396 is issued, indicating that an element or attribute is based on a union type but the data type does not correspond to any of the valid data types in the union.
- **Explanation:** When writing an element or attribute that is based on a union type, the MRM XML writer uses an xsi:type attribute, where present, to resolve the union.

  If an xsi:type attribute is not present, the writer tries to match the data type of the literal value in the tree to a simple type in the union. If the literal value cannot be matched to any of the simple types in the union, it reports a writing error.

- **Solution:** Perform one of the following steps:
  - Check that the message or ESQL contains a valid value for the element or attribute.
  - Check that the union type on which the element is based contains the correct list of simple types.
  - Consider using an xsi:type attribute to resolve the union explicitly.
  - Consider changing the type of the element in the tree to correspond with one of the union member types.

A union type contains two or more simple types that are derived from the same fundamental type

Procedure

- **Scenario:** A warning is issued during logical validation indicating that a union type contains two or more simple types that are derived from the same fundamental type.
- **Explanation:** The integration node does not apply value constraints until the data is in the logical tree. Therefore, it is not possible to choose between two simple types that are derived from the same fundamental type but with different constraints. For example, the integration node cannot differentiate between a member type of integer with a range of 1-10 and another member type of integer with a range of 11-20, therefore it resolves the union to the first member type of integer.
- **Solution:** Ensure that the union type that is identified in the warning does not contain more than one simple type that is derived from the same fundamental type or ensure that the ordering of such member types is as desired.

A list type is based on a union that also contains a list type

Procedure

- **Scenario:** An error message is issued during logical validation, indicating that a list type is based on a union type that includes a list type as one of its member types.
- **Explanation:** Lists of lists are not legal. An item type of a list type cannot be a list type itself nor can it be derived at any level from another list type. Therefore, a list type cannot have an item type of a union type that includes a list type as one of its member types.
- **Solution:** Ensure that any union type specified as an item type for a list type does not include a list type as one of its member types.

A union type contains an enumeration or pattern facet

Procedure

- **Scenario:** An error message is raised during logical validation, indicating that a union type contains an enumeration or pattern facet that is not supported because enumeration and pattern facets must be specified directly on the member type.
- **Explanation:** The integration node cannot support the union type facets of pattern and enumeration applied directly to a restriction of a union type. It can support facets directly on the chosen member type only. The Message Definition editor provides support for these facets to enable the import of schemas using direct facets on a restriction of a union type but it issues a warning notifying that they will be ignored by the integration node.
- **Solution:** If you want to use the enumeration or pattern facets, specify them directly on the member type and not on the union type itself.

Error message BIP5505 is issued because input data is not valid for the data type

Procedure

- **Scenario:** A data conversion fails while a message is being read or written. Error message BIP5505 is issued, indicating that the input data was not valid for the data type.
- **Solution:**

  Perform the following checks:

  – Ensure that the bit stream is correctly aligned to the model. If the incoming message is not consistent with the model, the wrong bytes are allocated to fields, and the data presented to a field is unlikely to contain valid data for the logical type.

  – Ensure that the correct encoding and CodedCharSetId are applied to the body of the input message. If the wrong encoding is used, the input message bit stream is subject to byte swapping and changes to the byte order. This change can affect the nature of data that applies to a field and make it unusable.

  – Ensure that an appropriate IBM MQ format is applied to the body of the input message. For example, if MQSTR is applied to a bit stream that contains numeric data, character conversions, rather than encodings, are used to translate the bytes. This use can change the meaning of bytes and make them not valid for the logical data type. If character conversions are used with negative external decimals, consider using the EBCDIC custom option.

  – If the error is on a numeric field, ensure that the input data is compatible with the physical format settings. For example, if the field is signed, ensure that a valid sign byte has been passed in, that it corresponds to the orientation of the sign, and that it is included or excluded as appropriate.

  – If the input message is found not to be compatible with the MRM message definitions, check the message at the source. If it is sent over channels, check that the conversion settings on the channels are correct.

A deprecation error is issued on an imported `.mrp` file

Procedure

- **Scenario:** You have imported an `.mrp` file, and get an error message indicating that complex elements or embedded simple types are deprecated.
- **Explanation:** Complex elements and embedded simple types have no exact equivalent in XML Schema. The closest equivalent to a complex type in XML Schema is to derive a complex type from a simple type. However, such a type is not allowed to contain elements; only attributes are allowed.
- **Solution:** If your complex type contains elements that cannot be modeled as attributes, set the **`Mixed content`** flag on the complex type. If you need to parse messages that contain mixed content, set the **`Mixed content`** flag on the parent complex type. The anonymous data that used to be modeled by the complex type or the embedded simple type, then appears as a self-defining node in the parsed message tree.

User trace detects an element length error

Procedure

- **Scenario:** You have defined an MRM message and when you execute the message flow, an error message is written to the user trace indicating that the length of an element is invalid.
- **Explanation:** You have added an element of type string to the complex type making up the message, but you have not defined the length of the string in the instance of the element in its type.

- **Solution:**
  a) Open the message definition file in the Message Definition editor.
  b) In the Outline view, expand the appropriate complex type, then click the element to display its properties in the editor area.
  c) In the Properties Hierarchy, make sure that the element is selected within the CWF physical format under **Physical Properties**.
  d) Specify the properties that this element will have when it is part of the complex type; for example, in the above case, make sure that the **Length Count** value is set.
  e) Save any changes in the Message Definition editor and redeploy the message.

MRM dateTime value has changed after it has been parsed

Procedure

- **Scenario:** You have defined an MRM dateTime element but the value created in the message tree is different from the value that you defined.
- **Explanation:** The parser uses lenient dateTime checking, converting out-of-band data values to the appropriate in-band value.
- **Solution:** See the information about lenient dateTime checking in Message Sets: DateTime as string data with particular reference to how years and fractional seconds are represented.

Resolving problems when using messages

Use the advice given here to help you to resolve common problems that can arise when you use messages.

About this task

Procedure

A communication error is issued when you use the enqueue facility

Procedure

- **Scenario:** You use the enqueue or dequeue tools to put a message on a queue, but an error message is issued indicating that a communication error has occurred with the queue manager name.
- **Explanation:** The IBM MQ queue manager has not started.
- **Solution:** Restart the IBM MQ queue manager.

The enqueue facility is not picking up changes made to a message

Procedure

- **Scenario:** You are using the IBM App Connect Enterprise Toolkit message enqueue facility to put messages to IBM MQ queues. You have updated a message and want to put the message to the queue, but your changes do not seem to have been picked up.
- **Solution:**
  a) Close, then reopen your enqueue file.
  b) Select the message that you want to put to the queue.
  c) Save and close the enqueue file.
  d) Select the menu next to the **Put a message to a queue** icon.
  e) Click **Put message**.
  f) Click the enqueue file in the menu.
  g) Click **Finish**.

  This action puts your updated message to the queue.

You do not know which header elements affect enqueue

Procedure

- **Scenario:** When using the Enqueue editor, the accounting token, correlation ID, group ID, and message ID in the message header do not seem to affect behavior.
- **Explanation:** These fields do not affect behavior because they are not serialized properly.

Enqueue message files are still listed after they have been deleted

Procedure

- **Scenario:** Enqueue message files are still listed in the menu after they have been deleted.
- **Explanation:** Deleted enqueue files are not removed from the menu. Nothing happens if you selecting these files.

The ESQL transform of an XML message gives unexpected results

Procedure

- **Scenario:** You have created an XML message that looks like the following content:

  ```
  <?xml version ="1.0" standalone="no"?><!DOCTYPE doc
  [<!ELEMENT doc (#PCDATA)*>]><doc><I1>100</I1></doc>
  ```

  You apply the ESQL transform:

  ```
  SET OutputRoot.XMLNS.doc.I1 = 112233;
  ```

  This transform generates the XML message (after serialization):

  ```
  <?xml version ="1.0" standalone="no"?><!DOCTYPE doc
  [<!ELEMENT doc (#PCDATA)*>]<I1>112233<I1>><doc><I1>100</I1></doc>
  ```

  The new value for *I1* has been put inside the DOCTYPE, and has not replaced the value of *100*, as you expected.
- **Explanation:** The XML in your message contains two doc elements:
  - The doctype element
  - The xmlElement that represents the body of the message

  The parser has found the first instance of an element called doc and has created a child *I1* with the value *112233*.
- **Solution:** To assign a new value to the element *I1* within the message body element doc, explicitly identify the second doc element, in the following way:

  ```
  SET OutputRoot.XMLNS.(XML.tag)doc.I1 = 112233;
  ```

An XML message loses carriage return characters

Procedure

- **Scenario:** You are parsing an input XML message that contains carriage return or line feed characters, or you are writing an output XML message that contains carriage return line feed characters in an XML element. However, some or all the carriage return characters are not present in the output message.
- **Explanation:** This behavior is correct, as described by the XML specification, and occurs in the XML, XMLNS, or XMLNSC domains.

  In XML, the main line separator characters is a line feed character. Carriage return characters are accepted in an XML document, but an XML parser normalizes any carriage return line feed characters into a single-line feed character. For more information, see the latest XML specification at Extensible Markup Language (XML), in particular, Section 2.11 *End of Line Handling*.

  You cannot work around this problem by embedding your data into a CDATA section; the XML specification states that a CDATA section is intended only to escape blocks of text that contain

characters that could be interpreted as markup. In addition, CDATA sections are not protected from the parser.

You cannot use the `xml:space` attribute to preserve carriage return line feed characters; the XML specification states that normalization of carriage return and line feed characters is done before any other processing is performed.

- **Solution:** XML-compliant data must not rely on the presence of carriage return line feed characters because in XML, the line feed character is only a line feed character and XML-compliant applications or data must be aware that the parser normalizes any carriage return line feed characters.

The integration node is unable to parse an XML message

Procedure

- **Scenario:** You receive a large XML file and wrap it in a SOAP envelope to be passed to a .NET web service. The integration node is unable to parse the XML message
- **Explanation:** The message that you receive is defined as UTF-8 but it contains UTF-16 characters, such as £. The presence of these characters causes a problem with the parser: the integration node is unable to parse the XML message because it contains an invalid character.
- **Solution:** Force the coded character set ID (CCSID) to 1208 instead of the default 437.

Error message BIP5004 is issued by the XMLNS parser

Procedure

- **Scenario:** Error message BIP5004 is issued, indicating that the XMLNS parser has encountered a problem with an input XML message.
- **Explanation:** This message is issued for a number of reasons. Some of the more commonly occurring scenarios when this message is issued are:
  - You have specified an invalid XML character in the input XML message.
  - You have included binary data in your XML message that, when treated as character data, is invalid.
  - The message has arrived as part of an IBM MQ message and the MQMD.CodedCharSetId does not correctly represent the XML text bit stream.
  - You have used characters that are recognized as markup.
- **Solution:**
  - Check that your sending application is sending only valid data. If, however, it is not possible to prevent invalid characters from being included in the XML message, represent it in BLOB domain and use the ESQL REPLACE function to replace or remove the invalid characters. You can then assign the modified bit stream to the required XML parser.

    In accordance with XML specification, a CDATA section can be used only to protect characters that would be interpreted as markup. It cannot be used to protect invalid characters or binary data from the XML parser.
  - If the input XML message contains binary data, ensure that the sending application is changed to represent this data as base encoded binary encoded data. If the application cannot be changed, represent the message in the BLOB domain, and extract and replace the binary data before the bit stream is assigned to the required XML parser.
  - Check that the incoming XML message is being represented by the correct MQMD.CodedCharSetId.

Error message BIP5378 is issued by the MRM parser

Procedure

- **Scenario:** Error message BIP5378 is issued, which reports a problem with a missing mandatory repeating element in an MRM message.
- **Explanation:** This message indicates that a mandatory repeating element is not present in the message.

  In previous releases, this condition was reported by error message BIP5374 which now reports only when a mandatory repeating element exists in the message but has the incorrect number of instances.

  If you have programmed automated error checking routines, review and change these routines if appropriate.
- **Solution:**

  Check your message definition to ensure that the element must be mandatory, and repeating. The error message tells you the elements that occur before and after the expected location of the missing element. If the definition is correct, the message has not been created correctly by the sending application, which must be amended.

Error message BIP5005 is issued by the Compute node

Procedure

- **Scenario:** You send a simple XML message into a simple message flow. The message is:

  ```
  <doc><I1>100</I1></doc>
  ```

  The Compute node in the message flow contains the following ESQL:

  ```
  SET OutputRoot.XMLNS.abc = InputBody;
  ```

  You expect the following output message to be created:

  ```
  <abc><doc><I1>100</I1></doc></abc>
  ```

  The Compute node generates error message BIP5005 and does not implement the ESQL.
- **Explanation:** You are assigning an element of one type (`root`) to an element of another type (`xmlElement`). The parser does not do this implicit cast for you.
- **Solution:** You can do the cast yourself in the ESQL to achieve the result that you want, using either of the following two casts:

  ```
  SET OutputRoot.XMLNS.(XML.Element)abc = InputBody;
  ```

  or:

  ```
  SET OutputRoot.XMLNS.(XML.tag)abc = InputBody;
  ```

A message is propagated to the Failure terminal of a TimeoutControl node

Procedure

- **Scenario:** The TimeoutControl node receives a message with invalid, corrupt, or missing timeout information. The message is propagated to the Failure terminal of the TimeoutControl node and an exception list is generated.

- **Explanation:** The following error conditions can cause propagation to the Failure terminal:
  - A timeout request has no Action or no Identifier.
  - A SET request has an Identifier that matches an existing stored SET request for this TimeoutControl node (identified by the Unique Identifier property) and AllowOverwrite of the original request is set to FALSE.
  - A CANCEL request has an Identifier that does not match an existing stored SET request for this TimeoutControl node (identified by the Unique Identifier property).
  - A SET request has a Count of 0 or is less than -1.
  - The StartDate or StartTime are not in the correct format (or one of the understood keywords).
  - The calculated timeout is in the past.
  - The Interval is less than 0, or 0 with a Count of -1.
- **Solution:** Check for one or more of the error conditions listed here, and correct them.

Message processing fails within a TimeoutNotification node

Procedure

- **Scenario:** A message is propagated to the Failure or Catch terminal of a TimeoutNotification node.
- **Explanation:** If the processing of a timeout generates an error within the TimeoutNotification node, an exception list is generated and a message is propagated to the Failure terminal. This action is done under the same transaction, if one is being used. If the Failure terminal is not connected, propagation does not occur.

  If an error happens downstream of the TimeoutNotification node after a successful propagation (either to the Out or Failure terminal), the message is propagated to the Catch terminal (all under the same transaction). If the Catch terminal is not connected, or the propagation along the Catch flow fails, the processing of that timeout is rolled back.
- **Solution:** Ensure that the Failure and Catch terminals of your TimeoutNotification node have been connected correctly.

An MRM CWF message is propagated to the Failure terminal

Procedure

- **Scenario:** Your MRM CWF message is propagated to a Failure terminal, and generates error messages BIP5285, BIP5125, and BIP5181 or messages BIP5285, BIP5125, and BIP5288.
- **Explanation:** These errors report an inconsistency between the length of the message being processed, and the length of the message that is defined in the message model.
- **Solution:** Ensure that the length of the message as defined in the CWF layer is accurate. Check and correct the definition.

Problems with XML attributes

About this task

XML tags are written where XML attributes are expected, and vice versa.

Procedure

- **Explanation:** The XML domains and the XML Wire Format in the MRM domain have their own representation of XML attributes.
  - XML domains rely on setting a field type of XML.Attribute in the message tree, because they have no model to parse against.
  - For the XML Wire Format in the MRM domain, the message model indicates whether an element is an attribute or a tag, therefore the message tree does not need to reflect whether a field is an attribute or a tag.

  Therefore, if fields are copied out of the XMLNS or MRM domains, the fact that fields are attributes is lost. This loss happens if they are copied out to each other, or to another message tree, such as the environment tree.

  This problem typically appears in the following situations:
- **Scenario 1:** You are writing an XML message in the MRM domain, and XML tags are being written instead of XML attributes.
- **Solution 1:** Check that your message tree has the same structure and sequence as the message model. If the message tree does not match the message model, the field is written as self-defining, and consequently the XML Render property is not used.
  - Switch on message validation. Validation shows where the message tree and message definition do not match.
  - Alternatively, take a user debug trace of the message flow; BIP5493W messages indicate which fields are being written as self-defining. Use this information to ensure that the message tree matches the model. When you have corrected any discrepancies, attributes are correctly written.
- **Scenario 2:** An MRM message has been copied to an XMLNS domain, and the XML attributes are now written as tags.
- **Solution 2:** Take one of these actions:
  - Serialize the XML message in the MRM domain, for example using the ESQL ASBITSTREAM function, then use the ESQL CREATE PARSE clause to reparse the message using the required XML domain.
  - When copying fields between the MRM domain and XMLNS, copy attribute fields individually, and specifically specify XML.Attribute on the target XML field.
- **Scenario 3:** An XML message has been copied to another message tree, such as Environment. When the message is copied back to the XML message tree, XML attributes are now seen as XML tags.
- **Solution 3:** Serialize the XML message, for example using the ESQL ASBITSTREAM function, then use the ESQL CREATE PARSE clause to reparse the XML message into the required target message tree. See CREATE statement for an example.
- **Scenario 4:** A portion of a non-XML message tree has been detached and attached to an XML tree, and XML tags are now written as XML attributes.
- **Solution 4:** Do not detach and attach portions of message trees that are owned by different parsers. Instead, use a tree copy.
- **Scenario 5:** An XML tag is copied to an XML attribute and the XML attribute is not written in the output message.
- **Solution 5:** When referencing the source XML tag, use the ESQL FIELDVALUE function to copy the specific field value to the target XML attribute field.

An MRM XML message exhibits unexpected behavior

Procedure

- **Scenario:** Your message flow is processing a message that you have modeled in the MRM. The message tree has not been created as you expected, the output XML message does not have the

expected contents, or the message contents are not being validated. No error message has been issued.

- **Explanation:** Two reasons might cause this problem:
  - **Explanation 1:** The XML physical format settings for a message set contain a property called Root Tag Name. This property defaults to MRM, in order to remain compatible with previous releases of the product. If you have not deleted the contents of this field, the MRM XMLNS parser expects the root tag for all XML messages to be MRM.

    Solution 1: Clear this field, or set it to the root tag used by all your XML messages. If you provide a value in this field, the root tag does not need to be modeled in all your message definitions.
  - **Explanation 2:** In order to remain backwards compatible, the integration node recognizes the format XML and invokes the XMLNS parser with specific default values. If you have created an XML physical layer for this message with the name XML, the integration node uses your definition. However, if you have not created an XML physical layer with this name, but have specified XMLNS as the format, either in the input node or the MQRFH2 header (when the input bit stream is parsed to a message tree), the integration node accepts the value specified and passes default values to the parser to create the message tree.

    Similarly, if you set XML in the Properties folder for the output message in the Compute node, this value is passed to the parser when it creates the message bit stream from the message tree, typically in the output node.

    The use of these default values by the parser might result in different content, structure, or both, for either message tree or output message. You can find further information about the action taken by the integration node in the user trace log where the following information might be written:

```
XMLWorker::initializeParse file:C:\s000\src\cpi\pwf\xml\xmlworker.cpp
line:126 message:5409.BIPmsgs
No dictionary present have you specified Wire Format 'XML' in error? ,
UserTrace   BIP5409E: XML Worker: Wire Format 'XML' specified.
                      Default MRM XML settings are being used because wire format
                      identifier 'XML' was specified and not found.
                      This can be due to an incorrect setting of the wire format
                      identifier in a message.
```

    Solution 2: If you have incorrectly entered the identifier of the format that you have defined, correct your code and try again. If you do not want the default action to be taken, define a physical layer that produces the required results.

The MRM parser has failed to parse a message because two attributes have the same name

Procedure

- **Scenario:** Two attributes in different name spaces have identical names. Error message BIP5117 is issued.
- **Explanation:** The MRM parser has failed to parse the message.
- **Solution:** Modify the attribute names so that they are not identical. This problem is a known limitation with the parser.

You encounter problems when messages contain EBCDIC newline characters

Procedure

- **Scenario:** If your bit stream input message contains EBCDIC newline (NL) characters, problems might arise if your message flow changes the target CCSID to an ASCII CCSID. For example, during conversion from CCSID 1047 to CCSID 437 (US PC ASCII), an NL character is translated from hex '15' to hex '7F', which is an undefined character. The error occurs because no corresponding code point for the newline character exists in the ASCII code page.

- **Solution:** You can overcome the problem in these cases:
  - On a system where the queue manager uses an ASCII code set, make sure that incoming messages do not contain any EBCDIC NL characters by:
    - Specifying that IBM MQ performs the conversion at the input node
    - Setting the queue manager attribute to convert NL to Line Feed (LF)
  - Where the input bit stream is character data, you can use MRM Tagged/Delimited message sets in a Compute node to replace the NL character with the required output.

The MIME parser produces a runtime error while parsing a message

Procedure

- **Scenario:** A MIME message is received by a message flow and produces a runtime error when the message is parsed.
- **Explanation:** The following errors can cause the MIME parser to reject a message:
  - The MIME header is not correctly formatted.
  - Either the top-level MIME header block, or a MIME header block for a nested multipart part, does not have a valid Content-Type header.
  - The top-level Content-Type has a media type of `message`.
  - The top-level Content-Type has a media type of `multipart` and no boundary definition.
  - A MIME header block is not correctly terminated by a blank line.
  - The constituent MIME parts are not correctly separated by boundary lines.
  - A boundary parameter value occurs in the content of a MIME part.
- **Solution:** Check the MIME message for one or more of the error conditions listed here, and correct them.

Runtime errors are issued when you write a MIME message from the logical message tree

Procedure

- **Scenario:** You are writing a MIME logical message tree as a bit stream and the parser generates a runtime error.
- **Explanation:** The following errors can cause the MIME parser to reject a logical message tree:
  - The root of the tree is not called MIME.
  - The last child of MIME is not called Parts or Data.
  - The Parts element has a value-only element, and this element is not the first or last child of Parts.
  - The Parts element has children that are not value-only elements or Part children.
  - The Parts element does not have any children called Part.
  - The last child of a Data element is not a BLOB.
- **Solution:** Check the MIME logical message tree for one or more of the error conditions listed here, and correct them.

Output message has an empty message body

Procedure

- **Scenario:** You have unexpectedly encountered an empty message body, or the ASBITSTREAM function has produced a zero length BLOB.

- **Explanation:** This error can happen for the following reasons:
  - You have created a message tree folder in a user-defined node but have not associated it with an owning parser. An owning parser is not associated with the message tree if you have created standard elements by using code similar or equivalent to:

    ```
    MbElement createElementAfter(int)
    MbElement createElementAfter(int, String, Object)
    MbElement createElementAsFirstChild(int)
    MbElement createElementAsFirstChild(int, String, Object)
    MbElement createElementAsLastChild(int)
    MbElement createElementAsLastChild(int, String, Object)
    MbElement createElementBefore(int)
    MbElement createElementBefore(int, String, Object)
    ```

  - You have used ESQL to create a message tree folder by using ESQL CREATE but without setting an owning parser for it. You might have used code similar or equivalent to:

    ```
    CALL CopyMessageHeaders();
       DECLARE outRef REFERENCE TO OutputRoot;
       CREATE LASTCHILD OF outRef AS outRef NAME 'BLOB';
       CREATE LASTCHILD OF outRef NAME 'BLOB' VALUE X'01';
    ```

    or the `outRef` reference variable was passed to an ESQL function or procedure that contained similar CREATE statements. You have not specified an owning parser by using the DOMAIN clause on the CREATE statement.
  - An MRM message tree has been constructed, then only part of it has been passed, by specifying a subfolder or field, to the ASBITSTREAM function with the parser mode option set to RootBitStream. This combination is not valid, and results in a zero length BLOB.
  - You have copied a message tree, or part of a message tree, to a folder and the owning parser association is not maintained.
- **Solution:** Depending on the reason for the empty message body or zero length BLOB, ensure that:
  - When you create a message tree folder in a user-defined node, associate an owning parser with it. Use code similar or equivalent to:

    ```
    createElementAfter(String parserName)
       createElementAsFirstChild(String parserName)
       createElementAsLastChild(String parserName)
       createElementBefore(String parserName)
    ```

  - When you use ESQL CREATE to create a message tree folder, use the DOMAIN clause to associate an owning parser with the message tree, for example:

    ```
    CALL CopyMessageHeaders();
       DECLARE outRef REFERENCE TO OutputRoot;
       CREATE LASTCHILD OF outRef AS outRef DOMAIN 'BLOB' NAME 'BLOB';
       CREATE LASTCHILD OF outRef NAME 'BLOB' VALUE X'01';
    ```

    This code creates a BLOB folder that has the BLOB parser associated with it.
  - When copying a message tree, or part of a message tree, ensure that the owning parser association is maintained, by first serializing, then reparsing the message into the appropriate message tree. An example scenario is where you have created a field:

    ```
    SET Environment.Variables.myMsg = InputRoot.XMLNS;
    ```

    Now you must pass it to the ASBITSTREAM function. Unless you use ESQL similar or equivalent to:

```
DECLARE xmlMsgBlob BLOB
          ASBITSTREAM(InputRoot.XMLNS, InputRoot.MQMD.Encoding, InputRoot.MQMD.CodedCharSetId);
  CREATE LASTCHILD OF Environment.Variables.myMsg DOMAIN('XMLNS')
         PARSE(xmlMsgBlob,
               InputRoot.MQMD.Encoding,
               InputRoot.MQMD.CodedCharSetId);
```

    the result is a zero length bit stream.

Output message has an invalid message body indicated by error message BIP5005, BIP5016, or BIP5017

Procedure

- **Scenario:** You have unexpectedly encountered a multi-root message body or a message without any root.
- **Explanation:** This error can occur when you have created an XML message tree folder with multiple roots or no root at all by:
  - Using a user-defined node
  - Using an MQGet node
  - Using ESQL
- **Solution:** Ensure that the final output message tree has one, and only one, XML root node.

Error message BIP5651 is issued when receiving a SOAP with Attachments message from a WebSphere Application Server client

Procedure

- **Scenario:** When a WebSphere Application Server client sends a SOAP with Attachments message to the integration node over JMS, error message BIP5651 is issued stating that no valid Content-Type header has been found.
- **Explanation:** When a WebSphere Application Server client sends a SOAP with Attachments message to the integration node over JMS, the MIME Content-Type value appears in the MQRFH2 header and not in the MIME message body.
- **Solution:** To solve this problem, the Content-Type value needs to be copied from the MQRFH2 header to the beginning of the message as a MIME header before the message is parsed. The following ESQL adds the Content-Type value to the beginning of the WebSphere Application Server message, then invokes the MIME parser on the result.

```
create procedure parseWAS_JMS(IN InputMessage reference,IN OutputMessage reference)
/**********************************************************************
* convert a WAS/JMS message to the correct format for the MIME parser
**********************************************************************/
begin
  -- get the data as a BLOB
  declare body BLOB InputMessage.BLOB.BLOB;

  -- get the Content-Type value from the RFH2 header. Content-Type is the only
  -- header which is critical for the MIME parser, but the same approach can be
  -- used for any MIME headers which have been stored under the RFH2 header.
  declare contentType char InputMessage.MQRFH2.usr.contentType;

  -- add the contentType to the bit stream as part of an RFC822 header block
  set body = cast(('Content-Type: '||contentType) as blob ccsid 819)||x'0d0a0d0a'||body;

  -- invoke MIME parser on the modified bit stream
  CREATE LASTCHILD OF OutputMessage DOMAIN('MIME') PARSE(body);
end;
```

A message flow can take in the JMS message in the BLOB domain, and call the procedure shown here from a Compute node. The procedure can be called by using the following ESQL from a Compute node:

```
CALL CopyMessageHeaders();                -- standard procedure to copy headers
CALL parseWAS_JMS(InputRoot, OutputRoot);  -- parse the ‘body' as MIME
```

WebSphere Application Server produces an error when receiving a SOAP with Attachments message

Procedure

- **Scenario:** When sending a SOAP with Attachments message to a WebSphere Application Server client using JMS, an error is generated stating that the bit stream contains unexpected characters.
- **Solution:** When the integration node sends a SOAP with Attachments message to WebSphere Application Server over JMS, the MIME Content-Type value must appear in the MQRFH2 header and not in the body of the message. The following procedure removes any MIME style headers from the front of the message bit stream and adds the Content-Type value to the MQRFH2 header. The supplied boundary value allows you to locate the start of the multipart MIME content.

```
create procedure writeWAS_JMS(IN OutputTree reference,IN boundary char)
/*****************************************************************************
* Serialise a MIME tree as normal, but then strip off any initial headers
* and save the Content-Type value in the RFH2 header as expected by WAS/JMS.
* Note: boundary - must be supplied with the leading hyphen pair
*****************************************************************************/
begin
  -- convert MIME subtree to BLOB
  declare body BLOB asbitstream(OutputTree.MIME);

  -- locate first occurrence of boundary and discard any data before this
  declare firstBoundary integer;
  set firstBoundary = position (cast(boundary as blob ccsid 819) in body);
  set body = substring(body from firstBoundary);

  -- save the MIME Content-Type value in the RFH2 header. Any other MIME
  -- headers which need to be preserved in the RFH2 header can be handled
  -- in the same way
  set OutputTree.MQRFH2.usr."contentType" = OutputTree.MIME."Content-Type";

  -- clear the MIME tree and create a new BLOB child for the modified body
  set OutputTree.MIME = null;
  CREATE LASTCHILD OF OutputTree DOMAIN('BLOB')PARSE(body);
end
```

Before calling this procedure, the message flow needs to be able to obtain the value of the boundary. This value might be available only within a Content-type header. The following procedure allows you to extract the Boundary value:

```
create procedure getBoundary(IN ct reference,OUT boundary char)
/*****************************************************************
* return value of the boundary parameter from a Content-Type value
*****************************************************************/
begin
  declare boundaryStart  integer;
  declare boundaryEnd    integer;

  set boundaryStart  = position('boundary=' in ct) + 9;
  set boundaryEnd    = position(';' in ct from boundaryStart);
  if (boundaryStart <> 0) then
    if (boundaryEnd <> 0) then
     set boundary  = substring(ct from boundaryStart for boundaryEnd-boundaryStart);
    else
      set boundary  = substring(ct from boundaryStart);
    end if;
  end if;
end;
```

A Compute node can invoke these procedures for sending a MIME message using the following ESQL:

```
SET OutputRoot = InputRoot;

declare boundary char;
CALL getBoundary(OutputRoot.Properties.ContentType, boundary);

CALL writeWAS_JMS(OutputRoot,boundary);
```

java_lang_StackOverflowError on AIX when processing a message flow that contains Java nodes and uses Java 5

Procedure

- **Scenario:** You get an abend on AIX when processing a message flow that contains Java nodes and uses Java 5.

  The abend file shows that there was an abend which indicates a segmentation fault, but a check of the `stderr` file shows a stack overflow in the JVM:

```
Exception in thread "Thread-15" java/lang/StackOverflowError: operating system stack overflow
        at com/ibm/broker/plugin/MbOutputTerminal._propagate    (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate    (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate    (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate (MbNode.java:1434)
        at com/ibm/broker/plugin/MbOutputTerminal._propagate   (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate   (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate   (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate   (MbNode.java:1434)
        at com/ibm/broker/plugin/MbOutputTerminal._propagate   (Native Method)
        at com/ibm/broker/plugin/MbOutputTerminal.propagate   (MbOutputTerminal.java:103)
        at com/ibm/xsl/mqsi/XMLTransformNode.evaluate   (XMLTransformNode.java:1002)
        at com/ibm/broker/plugin/MbNode.evaluate (MbNode.java:1434)
```

- **Explanation:**Java 5 has a parameter to adjust the stack size for Java threads. The default operating system stack size for Java 5 is only 256 KB. In certain message flows (for example, flows that contain Java user-defined nodes or XMLT nodes) this size might not be sufficient, and so you see a stack overflow error in the `stderr` file. Use the JVM option `-Xmso` to adjust the operating system stack for Java.

  You can display information about the stack by using the following command:

```
export MQSIJVERBOSE=-verbose:stack,sizes
```

  This command creates in the `stderr` file on startup an entry that has the folllowing content, or similar:

```
   -Xmca32K        RAM class segment increment
   -Xmco128K       ROM class segment increment
   -Xmns0K         initial new space size
   -Xmnx0K         maximum new space size
   -Xms125000K     initial memory size
   -Xmos125000K    initial old space size
   -Xmox250000K    maximum old space size
   -Xmx250000K     memory maximum
   -Xmr16K         remembered set size
   -Xlp0K          large page size
                   available large page sizes: 4K 16M
   -Xmso256K       OS thread stack size
   -Xiss2K         java thread stack initial size
   -Xssi16K        java thread stack increment
   -Xss256K        java thread stack maximum size
   -Xscmx16M       shared class cache size

Note: The stack size defaults to 256K.
```

- **Solution:**

  1. Issue the following command to set the operating system stack size for Java to 2 MB:

```
export IBM_JAVA_OPTIONS=-Xmso2m
```

  2. Restart the integration node.

Unable to access messages on a remote IBM MQ queue

Procedure

- **Scenario:** Error message BIP2677 occurs when an MQInput node is attempting to access messages on a remote queue.

- **Explanation:** This error can occur when an MQInput node is using the IBM MQ callback mechanism to receive messages from a queue.
- **Solution:** Check the IBM MQ completion and reason codes in the IBM MQ product documentation to determine the cause of the error. If the error is MQ_ENVIRONMENT_ERROR (2012), a likely cause is that the SHARECNV (shared conversations) parameter for the connected channel is set to `0`. To resolve the problem, ensure that the SHARECNV parameter for the channel is set to the default value of `10`.

An IBM MQ message does not complete backout processing, and `BackoutCount` increments continuously

Procedure

- **Scenario:** In a running message flow, an input message on an MQInput node is continuously backed out, and the `BackoutCount` increments indefinitely, even if the backout threshold was reached. BIP2630 appears in the local error log.
- **Explanation:** This error occurs when the message was backed out, and the queue manager for the integration node does not have a backout requeue queue or a dead letter queue defined, or there was an MQPUT error in connecting to either queue. The message cannot be put to a queue, and so is continuously backed out.
- **Solution:** If this situation occurs because neither a backout requeue queue or dead letter queue exists, define one (or both) of the queues to resolve the problem. If the condition that prevented the message from being processed is resolved, you can temporarily increase the value of the BOTHRESH attribute, which forces the message through normal processing.

Resolving problems when you are writing business rules

Use the advice that is given here to help you to resolve problems that can arise when you are writing business rules.

Resolving problems when you are writing business rules

Procedure

- **Scenario:** You are retrieving rules at run time from an IBM Operational Decision Manager repository but rules are not being updated automatically.
- **Explanation:** This problem can be caused when the TCP/IP port that you configure for IBM App Connect Enterprise to communicate with IBM Operational Decision Manager is used by another application.
  For example, the default IBM Operational Decision Manager TCP/IP port is 1883, which is also the default TCP/IP port for IBM MQ Telemetry.
- **Solution:** Check that the TCP/IP port that is configured for IBM Operational Decision Manager is not used by another application. If a conflict exists, change the port number for the JDBC Providers policy that defines the connection to the IBM Operational Decision Manager repository.

Resolving problems when you use the IBM App Connect Enterprise Toolkit

Use the advice given here to help you to resolve common problems that might occur you use the IBM App Connect Enterprise Toolkit.

About this task

Resolving problems that occur when connecting:

- "Your integration node is not recognized by the IBM App Connect Enterprise Toolkit" on page 2988
- "You cannot connect to the integration node from the IBM App Connect Enterprise Toolkit or command line" on page 2989

- “You cannot connect to a migrated integration node” on page 2989
- “Connection to the integration node is slow” on page 2990

Resolving error messages that occur when using the IBM App Connect Enterprise Toolkit:

- “IBM App Connect Enterprise Toolkit on Red Hat 7.n crashes with "JVM terminated. Exit code=160" error” on page 2991
- “After you start the IBM App Connect Enterprise Toolkit, runtime functions do not work” on page 2993
- “Errors occur while you are using the IBM App Connect Enterprise Toolkit” on page 2994
- “An error message is displayed when you start the Flow Exerciser” on page 2994
- “The plug-in repository cannot be found when you are installing a plug-in into the IBM App Connect Enterprise Toolkit” on page 2994
- “MSVCP120.dll missing message is issued ” on page 2995
- “An error is issued when you access the help system ” on page 2995
- “The IBM App Connect Enterprise Toolkit displays an error message after the error has been fixed” on page 2995
- “The message Unable to create part: filename.extension is issued” on page 2995

Resolving problems relating to the appearance of the workbench:

- “A view is missing from the workbench” on page 2996
- “You cannot close a message dialog box” on page 2996
- “You want to filter entries in the Problems view” on page 2996
- “Your message flow and message set projects have changed their appearance” on page 2996

Resolving non-specific problems when using the workbench:

- “Main menu entries missing on Linux” on page 2996
- “Editors do not update automatically when the same file is open in multiple windows” on page 2997
- “Deleting or closing a project takes a long time” on page 2997
- “You are experiencing poor performance when working with large or complex projects” on page 2997
- “You do not know how to return to the welcome page ” on page 2997

Resolving problems that occur when connecting the IBM App Connect Enterprise Toolkit and an integration node

Use the advice given here to help you to resolve common problems that can arise when you connect to an integration node.

Your integration node is not recognized by the IBM App Connect Enterprise Toolkit

Procedure

- **Scenario:** Your integration node is not recognized by the IBM App Connect Enterprise Toolkit.
- **Explanation:** Integration node names in the workbench are case sensitive.
- **Solution:** When you identify an integration node in the IBM App Connect Enterprise Toolkit, make sure that you use the same case that was used when it was created. On some operating systems, the name of the integration node might be folded to uppercase when it was created, even though you entered its name in lowercase.

You cannot connect to the integration node from the IBM App Connect Enterprise Toolkit or command line

Procedure

- **Scenario:** Error messages are issued when you try to connect to the integration node from the IBM App Connect Enterprise Toolkit or the command line.
- **Solution:** The following table shows the checks to carry out when an error message is issued in the IBM App Connect Enterprise Toolkit, or returned to the command line, when you try to connect to the integration node:

Error message		Actions
IBM App Connect Enterprise Toolkit	**Command line**	
BIP0914E	BIP1921E	Check that the integration node and its HTTP listener are running, and that the correct port is specified for the web user interface connection.
BIP0889E	BIP1920E	Check that the integration node is running. Check the system event log to ensure that the integration node is available for use, and correct all errors that are shown.
BIP0915E	BIP1923E	Check that the IBM App Connect Enterprise Toolkit local user ID is created on the integration node system. If it is not, create it. Check if the IBM App Connect Enterprise Toolkit local user ID that is created on the integration node system is authorized to connect to the web user interface.
	BIP1291E	Remove the **-q** parameter, and use the **-i** or **-p** parameters instead.

You cannot connect to a migrated integration node

Procedure

- **Scenario:** You cannot connect from the IBM App Connect Enterprise Toolkit, the IBM App Connect Enterprise Toolkit, or an IBM App Connect Enterprise Toolkit application to an integration node that you have migrated to Version 12.0.
- **Explanation:** The connection is failing because the IBM MQ channel does not exist.

  When you migrate an integration node to Version 12.0, the default server connection channel SYSTEM.BKR.CONFIG is created if it does not exist. However, if the integration node shared a queue manager with a Configuration Manager, this channel is deleted when you delete the Configuration Manager.
- **Solution:** Re-create the server connection channel SYSTEM.BKR.CONFIG on the queue manager.

You cannot connect to a remote integration node

Procedure

- **Scenario:** You have correctly configured your IBM App Connect Enterprise Toolkit to connect to a remote integration node, but attempting to connect to the remote integration node by using the

IBM App Connect Enterprise Toolkit results in error MQRC 2059. You can connect to your remote integration node from the same computer without error by completing the following steps:

a) Set the following environment variable:

- On Windows: `set MQSERVER=ChannelName/TCP/server-address(port)`
- On Linux: `export MQSERVER=ChannelName/TCP/'server-address(port)'`

b) Open an IBM App Connect Enterprise command prompt, and enter the following test command:

```
amqsputc queueName qmgrName
```

- **Explanation:** The network configuration of your IBM App Connect Enterprise Toolkit installation is incorrect.
- **Solution:**

a) In the IBM App Connect Enterprise Toolkit, click **Window** > **Preferences** > **General** > **Network Connection**.

The Preferences window opens, and the Network Connections preferences are displayed.

b) From the Active Provider list, select **Direct**.

For more information about the Active Provider setting, see the "Network Connections" topic in the Eclipse product documentation.

You can now connect to your remote integration node by using the IBM App Connect Enterprise Toolkit.

Connection to the integration node is slow

Procedure

- **Scenario:** Connection to the integration node is slow and you cannot complete other actions in the IBM App Connect Enterprise Toolkit when you request the following operations:
  - Creating an integration node
  - Creating an integration server
  - Deploying a BAR file, using the **Deploy a BAR File** wizard
  - Opening the Administration log editor
- **Solution:** Click **Cancel** on the progress monitor dialog box to cancel the in-progress connection.

  Connection to the integration node is canceled, so that you can perform other actions in the IBM App Connect Enterprise Toolkit. You can then resubmit the canceled operation.

Resolving errors that occur when using the IBM App Connect Enterprise Toolkit

Use the advice given here to help you to resolve common problems that can occur when you use the IBM App Connect Enterprise Toolkit.

About this task

- “Timeout issues and an inability to acquire a lock when you start the IBM App Connect Enterprise Toolkit” on page 2991
- “IBM App Connect Enterprise Toolkit on Red Hat 7.n crashes with "JVM terminated. Exit code=160" error” on page 2991
- “IBM App Connect Enterprise Toolkit on Linux becomes unresponsive after opening a DFDL or XSD file” on page 2992
- “Internal web browser in IBM App Connect Enterprise Toolkit does not work on Red Hat 7.n ” on page 2993

Timeout issues and an inability to acquire a lock when you start the IBM App Connect Enterprise Toolkit

Procedure

- **Scenario:** You try to start the IBM App Connect Enterprise Toolkit but it fails to start, or hangs on the splash screen. The log reports a timeout issue and an inability to acquire a lock.
- **Explanation:** The failure of the toolkit to start can be an issue with Windows defender on Windows 10. The Windows anti-malware takes away the CPU from the toolkit, making it impossible for the toolkit to start. You can look in the Windows task manager and observe that the *Antimalware Service executable* is using a similar amount of CPU as the toolkit.
- **Solution:** Add the IBM App Connect Enterprise installation directory to the exclusion list in Windows defender.

  1. Open **Windows Security**.
  2. Click **Virus and Threat Protection**.
  3. Click the **Manage Settings**.
  4. Click **Add or Remove Exclusions**.
  5. Click the "+" sign next to **Add an Exclusion**. The **Certificate** window is displayed.
  6. Select **Folder** from the menu to open **Windows Explorer**.
  7. Go to the installation directory for IBM App Connect Enterprise. For example, `C:\Program Files \IBM\ACE`.
  8. Click **Select Folder**. The **User Account Control** prompt is displayed.
  9. Select **Yes** in **User Account Control** to confirm your changes. The list of exclusions is updated to include your selection.

IBM App Connect Enterprise Toolkit on Red Hat 7.n crashes with "JVM terminated. Exit code=160" error

Procedure

- **Scenario:** When the IBM App Connect Enterprise Toolkit runs on Red Hat Linux 7.n, and you click the Patterns Explorer view or you use the Tutorials Gallery, the IBM App Connect Enterprise Toolkit can crash with the following error:

  ```
  JVM terminated. Exit code=160
  ```
- **Explanation:** Red Hat Linux 7.n ships with WebKitGTK version webkitgtk.x86_64 1.2.6-5.el6, which can have problems with the Eclipse SWT browser.

- **Solution:** To work around this problem, complete one of the following steps:
  - Update the IBM App Connect Enterprise Toolkit `eclipse.ini` file to use XULRunner instead of WebKitGTK for browser support by completing the following steps:
    1. Download XULRunner 10.0.4 ESR 64-bit from http://ftp.mozilla.org/pub/mozilla.org/xulrunner/releases/10.0.4esr/runtimes/xulrunner-10.0.4esr.en-US.linux-x86_64.tar.bz2.
    2. Expand the compressed file into a convenient directory. For example, `/usr/lib/xulrunner_new`.
    3. Add the following lines to the end of the `eclipse.ini` file in the `tools` directory under the IBM App Connect Enterprise installation directory:

       ```
       -Dorg.eclipse.swt.browser.XULRunnerPath=filepath/xulrunner
       -Dorg.eclipse.swt.browser.DefaultType=mozilla
       ```

       For example:

       ```
       -Dorg.eclipse.swt.browser.XULRunnerPath=/usr/lib/xulrunner_new/xulrunner
       -Dorg.eclipse.swt.browser.DefaultType=mozilla
       ```

    4. Restart the IBM App Connect Enterprise Toolkit.

IBM App Connect Enterprise Toolkit on Linux becomes unresponsive after opening a DFDL or XSD file

Procedure

- **Scenario:** When the IBM App Connect Enterprise Toolkit runs on Linux (for example, Red Hat Linux 7.n) and you open a DFDL or XSD file in the DFDL editor or XML Schema Editor, the IBM App Connect Enterprise Toolkit can become unresponsive and might not shut down correctly. Killing the `javaw` process manually closes the IBM App Connect Enterprise Toolkit, and might result in a `javacore` file with the main thread in the following method:

  ```
  org/eclipse/swt/internal/gtk/OS._gtk_enumerate_printers(Native Method)
  ```

- **Solution:** This issue is caused by Eclipse Bug 153936 (Eclipse freezes when opening an editor). Complete one of the following steps to resolve the problem:
  - Add the following line to the end of the `eclipse.ini` file in the tools directory under the IBM App Connect Enterprise installation directory:

    ```
    -Dorg.eclipse.swt.internal.gtk.disablePrinting
    ```

  - Install the GTK2 32-bit drivers by running the following command from the command line:

    ```
    sudo yum install gtk2.i686
    ```

Internal web browser in IBM App Connect Enterprise Toolkit does not work on Red Hat 7.n

Procedure

- **Scenario:** When the IBM App Connect Enterprise Toolkit runs on Red Hat Linux 7.n and you use any of the views that require an internal web browser, you might see one or more of the following error messages:
  - In the Pattern Specification editor:

    ```
    No internal web browser can be resolved. Pattern specifications is opened in default
     system web browser.
    ```

  - In the Tutorials Gallery view:

    ```
    No more handles [Unknown Mozilla path (MOZILLA_FIVE_HOME not set)]
    ```

  - In the XML View tab in the DFDL Test - Logical Instance view:

    ```
    The default browser installed on your system is not compatible or environment variables
     need to be specified.
    Consult the Welcome screen (Help and Welcome in the menu) for information about
     configuring your system.
    ```

- **Solution:** To work around this problem, update the IBM App Connect Enterprise Toolkit eclipse.ini file to use XULRunner instead of WebKitGTK for browser support, by completing the following steps:
  a) Download XULRunner 10.0.4 ESR 64-bit from http://ftp.mozilla.org/pub/mozilla.org/xulrunner/releases/10.0.4esr/runtimes/xulrunner-10.0.4esr.en-US.linux-x86_64.tar.bz2
  b) Expand the compressed file into a convenient directory (such as /usr/lib/xulrunner_new).
  c) Add the following lines to the end of the eclipse.ini file in the tools directory under the IBM App Connect Enterprise installation directory:

     ```
     -Dorg.eclipse.swt.browser.XULRunnerPath=<location of unzipped download file>/xulrunner
     -Dorg.eclipse.swt.browser.DefaultType=mozilla
     ```

     For example:

     ```
     -Dorg.eclipse.swt.browser.XULRunnerPath=/usr/lib/xulrunner_new/xulrunner
     -Dorg.eclipse.swt.browser.DefaultType=mozilla
     ```

  d) Restart the IBM App Connect Enterprise Toolkit.
  e) If the IBM App Connect Enterprise Toolkit fails with an error saying `JVM terminated. Exit code=160`, run the following command before launching the toolkit in the same terminal window:

     ```
     export LD_LIBRARY_PATH=<location of unzipped download file>/xulrunner:$LD_LIBRARY_PATH
     ```

     For example:

     ```
     export LD_LIBRARY_PATH=/usr/lib/xulrunner_new/xulrunner:$LD_LIBRARY_PATH
     ```

After you start the IBM App Connect Enterprise Toolkit, runtime functions do not work

Procedure

- **Scenario:** After you start the IBM App Connect Enterprise Toolkit, runtime functions including tutorials, patterns, the Flow Exerciser, and integration node administration do not work correctly.

- **Explanation:** This error occurs if you start the IBM App Connect Enterprise Toolkit by using one of the following commands to start Eclipse:
  - On Windows: `eclipse.exe`
  - On Linux: `./eclipse`
- **Solution:** Always start the IBM App Connect Enterprise Toolkit by using one of the following methods:
  - On Windows:
    - Use the Windows menu option.
    - From the IBM Integration Console, type `ace toolkit`.
  - On Linux:
    - From the command environment, type `./ace toolkit`.

Errors occur while you are using the IBM App Connect Enterprise Toolkit

Procedure

- **Scenario:** An error has occurred while you are using the IBM App Connect Enterprise Toolkit, and you want information to help you to diagnose the problem.
- **Solution:** Some errors that occur in the IBM App Connect Enterprise Toolkit are logged in the `.log` file in your `workspace\.metadata` directory.

  You can view this log by switching to the Plug-in Development perspective, and clicking the Error Log tab in the lower-right pane. The log shows in which plug-in the error occurred, and gives further information.

  You can also use trace to try and determine the cause of your problem. See "Using trace" on page 3026 for more information about tracing.

An error message is displayed when you start the Flow Exerciser

Procedure

- **Scenario:** Either your IBM App Connect Enterprise Toolkit is Version 10.0.0.1 (or later) and your integration node is Version 10.0.0.0, or your IBM App Connect Enterprise Toolkit is Version 10.0.0.0 and your integration node is Version 10.0.0.1 (or later). You open a message flow in the message flow editor and start the Flow Exerciser. The flow deploys correctly, but the progress window displays a failure message. The error contains the following text:

  ```
  BIP2210E: Invalid configuration message: attribute name 'testRecordMode' not valid for target
   object 'object_name'
  ```

- **Solution:** If the IBM App Connect Enterprise Toolkit is Version 10.0.0.0, you can use the Flow Exerciser only with an integration node that is Version 10.0.0.0. If the IBM App Connect Enterprise Toolkit is Version 10.0.0.1 (or later), you can use the Flow Exerciser only with an integration node that is Version 10.0.0.1 (or later).

  To check the version of an integration node within the IBM App Connect Enterprise Toolkit, click on the integration node in the **Integration Nodes** view, and then look in the **Properties** view.

Windows *The plug-in repository cannot be found when you are installing a plug-in into the IBM App Connect Enterprise Toolkit*

Procedure

- **Scenario:** You are installing a plug-in into the IBM App Connect Enterprise Toolkit and an error message is issued stating that no repository that contains the plug-in can be found.

- **Solution:** Ensure that you start the IBM App Connect Enterprise Toolkit by right clicking it in the **Start** menu and selecting **Run as administrator**.

Windows *MSVCP120.dll missing message is issued*

Procedure

- **Scenario:** You are using the IBM App Connect Enterprise Toolkit and a pop-up window is seen that states `mqsisetsecurity.exe has stopped working`. Clicking the details in the pop-up window shows that the MSVCP120.dll file is missing.
- **Solution:** Download and install the Microsoft Visual C++ 2010 Redistributable Package.

An error is issued when you access the help system

Procedure

- **Scenario:** You are accessing the help system through the IBM App Connect Enterprise Toolkit, and an error message is issued stating that the Web page that you requested is not available offline.
- **Explanation:** This error might occur if you previously had a connection to a Web-based version of the help system and lost it, or if you have **`Work Offline`** selected in your Internet Explorer options.
- **Solution:** Click **Connect** to load the help system.

The IBM App Connect Enterprise Toolkit displays an error message after the error has been fixed

Procedure

- **Scenario:** The IBM App Connect Enterprise Toolkit is in an inconsistent state, or displays an error message after the error has been fixed
- **Solution:** Clean the workspace:
  1. From the IBM App Connect Enterprise Toolkit, click **Project** > **Clean**.
  2. Click **Clean all projects** and **Finish**.

     This action cleans the whole workspace of any internal files, which are then re-created so that none of your data is lost.

The message `Unable to create part: filename.extension` is issued

Procedure

- **Scenario:** You open an editor in the IBM App Connect Enterprise Toolkit, and an error message is raised stating that a file cannot be created.

  This problem is caused by one of following situations:
- **Explanation 1:** The given file is not associated with a recognized editor.
  - **Solution 1:** Right-click the file and choose the default editor to open it, or choose another editor if the default editor cannot open it.
- **Explanation 2:** The given file contains syntax errors and cannot be loaded into the chosen editor. However, if you then try to open a valid file, you get the same error message repeatedly.
  - **Solution 2:** Restart the IBM App Connect Enterprise Toolkit; do not load the file into an editor again until you have fixed the syntax errors.

Resolving problems relating to the appearance of IBM App Connect Enterprise Toolkit

Use the advice given here to help you to resolve common problems that relate to the appearance of the IBM App Connect Enterprise Toolkit.

A view is missing from the workbench

Procedure

- **Scenario:** A view you are expecting to be displayed in IBM App Connect Enterprise Toolkit is not visible, or you have closed a view.
- **Explanation:** The perspective has changed since it was first created, and the default views are no longer visible.
- **Solution:** To restore the default views in a perspective you can reset the perspective. Click **Window** > **Reset perspective**.

You cannot close a message dialog box

Procedure

- **Scenario:** You have opened a menu, and a message dialog box tells you that previous changes made have been processed successfully by the integration node. You cannot close the dialog.
- **Solution:** If you cannot close the dialog box by clicking **OK**, press Esc.

You want to filter entries in the Problems view

Procedure

- **Scenario:** The Problems view has many entries, and it is difficult to find the entry that you want.
- **Solution:** At the top of the Problems View, click the icon with the arrows pointing to the right. This action opens a dialog box, in which you can tailor your selections to display only a subset of the entries in the view. For example, you can select only those entries for the project that you have selected.

Your message flow and message set projects have changed their appearance

Procedure

- **Scenario:** You have created a new simple project in the IBM App Connect Enterprise Toolkit and now all your message flow and message set projects look different.
- **Explanation:** When you create a new simple project, the IBM App Connect Enterprise Toolkit switches automatically to the Integration Development perspective.
- **Solution:** To return to the previous view, switch to the perspective with which you were working..

Resolving non-specific problems when using the IBM App Connect Enterprise Toolkit

Use the advice given here to help you to resolve some common problems that can occur when you use the IBM App Connect Enterprise Toolkit that are not dealt with in previous categories.

Main menu entries missing on Linux

Procedure

- **Scenario:** The main menu on Linux no longer contains items that relate to the IBM App Connect Enterprise Toolkit.

- **Explanation:** If you have installed the IBM App Connect Enterprise Toolkit in more than one package group, and have now removed one of those installations, a known restriction causes the main menu items to be removed when the component is uninstalled.
- **Solution:** Invoke the IBM App Connect Enterprise Toolkit from the command line.

  Navigate to the installation directory for the package group in which you have installed the IBM App Connect Enterprise Toolkit, and enter the following command:

  ```
  ./eclipse -product com.ibm.etools.msgbroker.tooling.ide
  ```

Editors do not update automatically when the same file is open in multiple windows

Procedure

- **Scenario:** You are working in the Integration Development perspective, and are using the associated editor to work with one or more resources; for example, you are editing a message flow in the message flow editor or an ESQL module in the ESQL editor.

  You have clicked **Window** > **New Window** to create a second Eclipse view, and have opened the same resource in the second window. Changes that you make to the resource in the first editor window are not reflected in the second editor window.
- **Explanation:** The IBM App Connect Enterprise Toolkit editors do not automatically update multiple windows in which you have opened the same resource.
- **Solution:** Save the contents of the resource file in the first editor window, then close and reopen additional windows.

  The reopened windows reflect the updated content.

Deleting or closing a project takes a long time

Procedure

- **Scenario:** Deleting or closing a project to save memory takes a long time.
- **Explanation:** If a project is referenced by other projects, removing that project requires all the other projects, and the projects that refer to them recursively, to be built fully. This process occurs to keep the content-assist and validation models current.
- **Solution:** To keep a project open in the workspace requires very little memory, therefore you do not need to close or delete projects.

You are experiencing poor performance when working with large or complex projects

Procedure

- **Scenario:** You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large or complex projects.
- **Explanation:** Frequent project changes, such as adding and removing projects, or using **Project** > **Clean**, use large amounts of memory because of the size and number of files and the connections between them.
- **Solution:** Increase your system memory.

You do not know how to return to the welcome page

Procedure

- **Scenario:** You do not know how to return to the welcome page that was displayed in the IBM App Connect Enterprise Toolkit when you first started using it.

- **Solution:** To open the welcome page:
  a) From the **Help** menu, select **Welcome**.
     If only one welcome page is available, it is displayed. If more than one is available, a list is displayed.
  b) Select the welcome page that you want, for example, IBM App Connect Enterprise IBM App Connect Enterprise Toolkit.
  c) Click **OK**.

Resolving problems when using Data Analysis

Use the following advice to help you to resolve problems that can arise when you use Data Analysis.

About this task

- "'DOCID' and 'ID' columns are inserted into my database" on page 2998

'DOCID' and 'ID' columns are inserted into my database

Procedure

- **Scenario:** You map your database to Data Analysis and the columns 'DOCID' and 'ID' are inserted into your database.
- **Explanation:** The 'DOCID' and 'ID' columns are added to provide a unique key string for database operations. The 'DOCID' column is set to a unique string that you can use to identify the insertion of each entry into the database and remove if necessary. The 'ID' is the foreign key that links related tables together.
- **Solution:** Works as designed.

Resolving problems when using databases

Use the advice given here to help you to resolve problems that can arise when using databases.

Before you begin

- Read "Is there a problem with a database?" on page 2885

Procedure

- "DB2 error message SQL0443N is issued" on page 2999
- "DB2 error message SQL0805N is issued" on page 2999
- "DB2 error message SQL0998N is issued on Linux" on page 3000
- "DB2 error message SQL0998N or SQL1248N is issued" on page 3000
- "DB2 error message SQL1040N is issued" on page 3000
- "DB2 error message SQL1224N is issued when you connect to DB2" on page 3000
- "You do not know how many database connections an integration node requires" on page 3002
- "You want to use XA with DB2 databases" on page 3002
- "The db2swit file that is needed for XA with DB2 databases cannot be loaded" on page 3002
- "XA coordination fails if the database restarts while the integration node is running" on page 3003
- "Error message BIP2322 IM004 is issued when you connect to an Informix database" on page 3003
- "On Oracle, a database operation fails to return any rows, even though the rows exist" on page 3004
- "Integration node commands fail when the Oracle 10g Release 2 client runs on Linux on POWER with Red Hat Enterprise Linux Advanced Server V4.0" on page 3004

DB2 error message SQL0443N is issued

Procedure

- **Scenario:** After you upgrade your DB2 server to a new fix pack level, a DB2 error message SQL0443N is issued if you invoke a DB2 Call Level Interface (CLI) catalog function, such as `SQLTables()`, `SQLColumns()`, or `SQLStatistics()`.
  An example of the error message is:

  ```
  SQL0443N Routine "SYSIBM.SQLTABLES" (specific name "TABLES") has returned an error SQLSTATE with diagnostic text SYSIBM:CLI:-805". SQLSTATE=38553.
  ```

- **Solution:** Bind the `db2schema.bnd` file against each database by entering the following commands at a command prompt:

  ```
  db2 terminate
  db2 connect to database-name
  db2 bind path\db2schema.bnd blocking all grant public sqlerror continue
  db2 terminate
  ```

  where *database-name* is the name of the database to which the utilities must be bound, and *path* is the full path name of the directory where the bind files are located. For example, the default location on Windows is `C:\Program Files\IBM\SQLLIB\bnd\`.

  To list all the names of databases for a particular DB2 instance, run the DB2 CLI command **`db2 list database directory`**. For further information, see the DB2 documentation.

DB2 error message SQL0805N is issued

Procedure

- **Scenario:** When a message flow that includes a Database node runs, SQL error `SQL0805N NULLID.SQLLF000` is issued.
- **Solution:** Open a DB2 Command Line Processor window and issue a bind command to the database.

  Linux UNIX On Linux and UNIX systems, enter the commands:

  ```
  connect to db
  bind ~/sqllib/bnd/@db2cli.lst grant public CLIPKG 5
  connect reset
  ```

  where *db* is the database name.

  Windows On Windows systems, enter the commands:

  ```
  connect to db
  bind x:\sqllib\bnd\@db2cli.lst blocking all grant public
  ```

```
connect reset
```

where *x:* identifies the drive onto which you installed DB2, and *db* is the database name.

DB2 error message SQL0998N is issued on Linux

Procedure

- **Scenario:** You are trying to use a globally coordinated message flow with DB2 on Linux and error message SQL0998N is issued with Reason Code 09 and Subcode " ".
- **Solution:** Check that the LD_ASSUME_KERNEL environment variable is not set.

  If it is set, use the **unset** command to remove it from your environment and ensure that you modify your profile scripts so that it remains unset.

DB2 error message SQL0998N or SQL1248N is issued

Procedure

- **Scenario:** When you try to use a globally coordinated message flow with a DB2 database, you get one of the following error messages:
  - SQL0998N with Reason Code 09 and Subcode ""
  - SQL1248N with a message indicating that the database is not defined with the transaction manager
- **Solution:** Use the instructions in “Configuring global coordination with DB2” on page 709 to configure the database and XAResourceManager stanza.

DB2 error message SQL1040N is issued

Procedure

- **Scenario:** You are using a DB2 database, and error message BIP2322 is issued with error SQL1040N.
- **Explanation:** The following DB2 message indicates that the value of the DB2 database configuration parameter **maxappls** has been reached:

```
"SQL1040N The maximum number of applications is already connected to the database.
SQLSTATE=57030"
```

  DB2 has rejected the attempt to connect.
- **Solution:**
  a) Stop all integration nodes that connect to the affected database.
  b) Increase the value of the **maxappls** configuration parameter.

     Also, check the value of the associated parameter **maxagents**, and increase it in line with **maxappls**.
  c) Restart the DB2 database.
  d) Restart the integration nodes.

DB2 error message SQL1224N is issued when you connect to DB2

Procedure

- **Scenario:** DB2 error message SQL1224N is issued when you connect to a DB2 database. This error indicates that a database agent could not be started, or was ended because of a database shutdown or force command.

- **Solution:** On AIX, use TCP/IP to connect to DB2 databases, to avoid the shared memory limit of 10 connections.

  To set up AIX and DB2 loop-back to use a TCP/IP connection:

  a) Configure DB2 to use TCP/IP, and to start the TCP/IP listener.

     On the database server machine, log in as the DB2 instance owner, typically `db2inst1`, and issue the following commands:

     ```
     db2set DB2COMM=tcpip
     db2stop
     db2start
     ```

  b) If the DB2 connection port is not defined in `/etc/services`, edit the services file to add the DB2 connection and interrupt ports. You must use unique names, and port numbers that are not already defined in the services file; for example:

     ```
     db2svc1     3700/tcp          # DB2 Connection Service
     db2isvc1    3701/tcp          # DB2 Interrupt Service
     ```

  c) Update the DB2 configuration; for example:

     ```
     db2 update dbm cfg using svcename db2svc1
     ```

     where *db2svc1* is the name of the DB2 Connection port service in `/etc/services`.

     Alternatively, you can specify a port number directly.

  d) Stop and restart the database by using the following commands:

     ```
     db2stop
     db2start
     ```

  e) Catalog a new TCP/IP connection node:

     ```
     db2 catalog tcpip node NODENAME remote HOSTNAME server db2svc1
     ```

     where:

     NODENAME
     is the name of the new TCP/IP connection node. You can use `local` as your node name, if it is a unique identifier.

     HOSTNAME
     is the name of your computer.

     db2svc1
     is the name of the DB2 connection port service in `/etc/services`.

     Message DB20000I is displayed when the command completes successfully.

  f) Catalog the database with a new alias name; for example:

     ```
     db2 catalog database DATABASE as DBALIAS at node NODENAME
     ```

     where:

     DATABASE
     is the physical name of the database.

     DBALIAS
     is the database alias name that you want to use.

     Specify the new alias name in all subsequent references to the local database.

  g) Stop and start DB2:

     ```
     db2 terminate
     db2stop
     ```

```
db2start
```

h) Log on with the user ID under which the integration node is running.

i) Update the ODBC configuration file for each integration node to add definitions for the database:

a. At the top of the file, add a definition for the database alias name:

```
DBALIAS=IBM DB2 ODBC Driver
```

b. Add a new stanza for the database alias:

```
 [DBALIAS]
Driver=INSTHOME/sqllib/lib/db2o.o
Description=Database Alias
Database=DBALIAS
```

where *INSTHOME* is the path to your DB2 Instance directory.

j) Update your message flows to specify the alias database name, redeploy the BAR file to the integration node, and test the flows.

You do not know how many database connections an integration node requires

Procedure

- **Scenario:** You do not know how many database connections to set up for your integration node.
- **Solution:** Determine the number of database connections required by an integration node for capacity and resource planning. On DB2, the default action taken is to limit the number of concurrent connections to a database to the value of the **maxappls** configuration parameter; the default value for **maxappls** is 40. The associated parameter **maxagents** also affects the current connections.

  The connection requirements for a single integration node are:

  – Five are required by internal integration node threads.

  – One is required for each database access node to separate ODBC data source names for each message flow thread (that is, if the same DSN is used by a different node, the same connection is used).

You want to use XA with DB2 databases

Procedure

- **Scenario:** You want to use XA with one or more DB2 databases.
- **Solution:** Ensure that your queue manager is configured to use **ThreadOfControl=THREAD**.

  – Linux UNIX On Linux and UNIX systems, configure this parameter in the XAResourceManager stanza in the file `qm.ini` for the integration node queue manager.

  – Windows On Windows systems, configure this parameter in IBM MQ Explorer.

The `db2swit` file that is needed for XA with DB2 databases cannot be loaded

Procedure

- **Scenario:** Error messages are present in the IBM MQ queue manager error logs indicating that the `db2swit` file cannot be loaded.
- **Explanation:** IBM App Connect Enterprise provides a prebuilt switch load file that is used for XA with DB2. If this file cannot be loaded, XA transactions cannot take place.

- **Solution:** There are 2 reasons why this situation might arise:
  1. IBM App Connect Enterprise and IBM MQ might not have been correctly configured. To establish correct configuration, refer to the instructions in "Configuring global coordination with DB2" on page 709.
  2. The switch load file, `db2swit`, that is provided with IBM App Connect Enterprise might not be compatible with the version of DB2 that you are running. If this is the case, you must build your own version of the `db2swit` file. For instructions on building your own `db2swit` file, refer to the `sample/xatm/readme.db2` file in your IBM App Connect Enterprise installation.

XA coordination fails if the database restarts while the integration node is running

Procedure

- **Scenario:** XA global coordination fails, and you get an error like the following example, which is from a DB2 database:

```
Database error: SQL State '40003'; Native Error Code '-900';  Error Text '[IBM]
[CLI Driver] SQL0900N  The application state  is in error. A database connection
does not exist.SQLSTATE=08003'.
```

- **Explanation:** A globally coordinated message flow cannot automatically reconnect to a database if the database is restarted while the integration node is still running.
- **Solution:** Stop and restart the integration node if the database goes down, or is brought down for a scheduled maintenance.

Error message BIP2322 IM004 is issued when you connect to an Informix database

Procedure

- **Scenario:** You are using the **mqsicvp** command to connect to an Informix database, and you see the following error message.

```
BIP2322E: Database error: SQL State 'IM004'; Native Error Code '0'; Error Text '[DataDirect]
[ODBC lib]
Driver's SQLAllocHandle on SQL_HANDLE_ENV failed'. The error has the following diagnostic
 information:
SQL State 'IM004' SQL Native Error Code '0' SQL Error Text '[DataDirect][ODBC lib] Driver's
 SQLAllocHandle
on SQL_HANDLE_ENV failed'.
```

- **Explanation:** This error can be caused when the environment for the database has not been initialized.
- **Solution:** Check the documentation for the client on your integration node system for details of the actions that you must take. For example, you might have to specify the following environment variables:

```
export INFORMIXDIR=/installation_directory_of_informix_client_software
export PATH=${INFORMIXDIR}/bin:${PATH}
export INFORMIXSERVER=server_name
export INFORMIXSQLHOSTS=${INFORMIXDIR}/etc/sqlhosts
export TERMCAP=${INFORMIXDIR}/etc/termcap
export TERM=vt100
export LIBPATH=${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:
```

```
                ${INFORMIXDIR}/lib/cli:$LIBPATH
```

where *server_name* is defined in the file `sqlhosts` (the required value is typically the machine name), and the location of the file `sqlhosts` is set up as part of the installation process.

To configure your system to run this setup at the start of every session, add these statements to the login profile of the user that is going to run the integration node.

For more information, see "Running database setup scripts before starting an integration node" on page 83.

On Oracle, a database operation fails to return any rows, even though the rows exist

Procedure

- **Scenario:** You are using Oracle databases in your message flows, and ESQL binds against columns that are declared as data type CHAR, and those parameter markers are referenced in a WHERE clause. The database operation fails to return any rows, even though the rows exist.
- **Explanation:** Fixed-length character strings must be padded with blank characters on Oracle for this type of comparison to succeed.
- **Solution:** Define the CHAR columns as VARCHAR2 columns, or pad the ESQL variable with blank characters to the required column length, so that the comparison locates the required rows from the table.

Integration node commands fail when the Oracle 10g Release 2 client runs on Linux on POWER with Red Hat Enterprise Linux Advanced Server V4.0

Procedure

- **Scenario:** Integration node commands fail in an environment where an Oracle 10g Release 2 client runs on Linux on POWER with Red Hat Enterprise Linux Advanced Server V4.0.

  Abend files might be created, with contents like the following data:

  ```
  /opt/mqsi/lib/libCommonServices.so(.ImbAbendSignalHandler+0x10)[0x800017d3c0]
  [0x80022b33f0]
  /lib64/tls/libpthread.so.0[0x80cc2339d8]
  /lib64/tls/libpthread.so.0[0x80cc230e4c]
  /lib64/tls/libpthread.so.0[0x80cc22af54]
  bipservice[0x10005e38]
  /lib64/tls/libpthread.so.0[0x80cc22a1d8]
  /lib64/tls/libc.so.6[0x80cc0dd3e4]
  ```

  or

  ```
  /opt/mqsi/lib/libImbCmdLib.so(._ZN15ImbCreateTables15createAllTablesEv+0x38)
  [0x800014c9bc]mqsicreatebroker[0x10023324]mqsicreatebroker[0x1002c0c4]
  /opt/mqsi/lib/libImbCmdLib.so(._ZN10ImbCmdBase20processCommandStringEv+0x6c)
  [0x80001dcad4]mqsicreatebroker[0x1000d728]/lib64/tls/libc.so.6[0x80cc02423c]
  /lib64/tls/libc.so.6[0x80cc0243c4]
  ```

- **Explanation:** The Oracle installation library contains copies of `libgcc` library files. The Oracle user profile adds the directory containing these files to the environment variable LD_LIBRARY_PATH. This action causes the `libgcc` library files to be found before the system libraries, and leads to the failure of integration node commands and the production of abend files.
- **Solution:** Ensure that you add `/lib64` to the environment variable LD_LIBRARY_PATH before the Oracle library path. Before you run **`mqsiprofile`**, include a string like:

  ```
  /lib64::/usr/lib:/oracle/app/oracle/product/10.2.0/db_1/lib
  ```

  which shows `/lib64` preceding the Oracle library directory.

Error message BIP2322 `Driver not capable` is issued when you use an Informix database

Procedure

- **Scenario:** When you try to access an Informix database from a node in a message flow, the following error message is issued:

```
BIP2322E: Database error: SQL State 'HYC00'; Native Error Code '-11092';
Error Text '[Informix][Informix ODBC Driver]Driver not capable.'.
```

- **Explanation:** The integration node uses transaction statements, therefore the database must be created, and configured to enable logging.
- **Solution:** Consult your database administrator to ensure that transaction logging has been enabled on the Informix database that the integration node is trying to access.

  For example, create the database with a buffered log:

```
create database with [buffered] log;
```

Database updates are not committed as expected

Procedure

- **Scenario:** You have included a Database node, Compute node, or Filter node in a message flow, and you have set the **`Transaction`** property to Commit. The message flow has raised an exception and has rolled back, but the database updates have not been committed.
- **Explanation:** When you set **`Transaction`** to Commit, the database updates performed by the node are committed when the node completes successfully. If an exception is raised before the node has completed, and causes the message flow to be rolled back, the commit is not issued and the database updates are also rolled back. The conditions under which this situation can occur are:
  - The node itself causes an exception to be raised. The commit is never performed.
  - The ESQL contains a PROPAGATE statement. This statement does not complete until all processing along the path taken by the propagated message has completed, and control returns to the node. Only then can the commit be performed. If an exception is raised along this path, control is not returned and the database updates are rolled back as part of the message flow.
- **Solution:** Review the operation of the node that performs the database updates.
  For example, you might be able to split the work between two nodes, with the first updating the database, and the second propagating the output message. Consider changing the ESQL code to process the message in a different way.

You want to list the database connections that the integration node holds

Procedure

- **Scenario:** You want to list the database connections that the integration node holds.
- **Solution:** The integration node does not have any functionality to list the connections that it has to a database. Use the facilities that your database supplies to list connections. Refer to the documentation for your database to find out how to perform this task.

You want to know whether the password that is set for a database is as expected

Procedure

- **Scenario:** You want to check the password that is set for a database that is associated with an integration node is the password that you expect.
- **Solution:** Use the **mqsireportdbparms** command with the integration node name, user ID, and password. The command reports whether the entered password was correct or not. For more information, see "Checking the password for a resource that is used by an integration server" on page 2630.

The queue manager finds the XA resource manager is unavailable when configured for XA with DB2 on Windows

Procedure

- **Scenario:** You have configured a queue manager for XA coordination with DB2 on your Windows computer.

  When you restart the queue manager, it reports error AMQ7604 in the queue manager error log. All subsequent attempts at XA coordination between IBM MQ and DB2 fail.

  The error message has the following content, or similar:

```
23/09/2008 15:43:54 - Process(5508.1) User(MUSR_MQADMIN) Program(amqzxma0.exe)
AMQ7604: The XA resource manager 'DB2 MQBankDB database' was not available
when called for xa_open.
The queue manager is continuing without this resource manager.
```

- **Explanation:** The user ID that runs the IBM MQ Services process `amqmsrvn.exe`, which has a default value of MUSR_MQADMIN, is running with an access token that does not contain group membership information for the group DB2USERS.
- **Solution:** Check that the IBM MQ Services user ID is a member of the group DB2USERS, stop the IBM MQ service (for example, by issuing the command **net stop "IBM MQSeries"**), and all other processes that are running under the same user ID, and then restart these processes.

  You can restart your computer to stop and restart these processes after you have checked the user ID status, but this action is typically not required.

Error messages are received when you are trying to remove a DB2 database on Windows when you are using a sample

Procedure

- **Scenario:** You are running a sample, and you are trying to remove a DB2 database on your Windows computer, and you receive a BIP9835E error message with the error code SQLSTATE=57019.

  The error message has content like the following data:

```
BIP9830I: Deleting the DB2 Database <Your database name>.
BIP9835E: The DB2 batch command failed with the error code SQLSTATE=57019.
The database <Your database name> could not be created/deleted.
The error code SQLSTATE=57019 was returned from the DB2 batch command.
```

- **Explanation:** If you use the DB2 Control Center to perform a query, a connection is opened against the database. This connection stays open until the DB2 Control Center is closed, at which point the connection is ended.
- **Solution:** Close the DB2 Control Center application, and try to run the sample again.

DB2 error message SQL7008N is issued

Procedure

- **Scenario:** You are using DB2 and encounter error SQL7008N when updating or inserting into tables on iSeries.
- **Explanation:** SQL7008N is a common error when the tables being accessed on an iSeries server are not being journaled.
- **Solution:** To correct the error, take one of the following steps:
  - Journal your tables.
  - Change the isolation level to `No Commit`. You can change the isolation level of the database alias referenced by your ODBC data source to `No Commit` by using the following DB2 command:

    ```
    db2 update cli cfg for section <db_alias> using TxnIsolation 32
    ```

Resolving problems when using publish/subscribe

Use the advice given here to help you to resolve common problems that can occur when you run publish/subscribe applications.

About this task

Procedure

- "Application responses are not received" on page 3007
- "Your application is not receiving publications" on page 3008
- "Publishing a message causes a filter error" on page 3008
- "Symbols in subscription filters cause problems" on page 3009
- "There are performance problems on AIX when the JIT compiler is not loaded" on page 3009
- "The Publication node fails with MQRC 2035" on page 3009

Application responses are not received

Procedure

- **Scenario:** Application responses are not received.
- **Explanation:** Depending on the application code, a publisher or subscriber might request confirmation that its message was processed successfully. Responses can make debugging client problems much easier, because a response code is given if a problem occurs.

  Typically, a response is always returned to a subscriber. However, for the publishing side, a message might be published without the knowledge of the originating application (for example, by using the default topic property on the input node, or by using a Compute node to modify this property in a message flow). The results of processing that message are still logged in user trace, which might give clues as to what is happening.

  If a response is not received, it is typically due to one of the following causes:

  - The system is busy. Messages might build up on the input queue, and the client might not be waiting long enough for its response.
  - IBM MQ expiry is being used. There are cases where this option is what is required, but the expiry of the input message is copied to the response. As a result, the message might expire on the input

queue, or it might expire on the way back to the client. This situation is not an error, even with a persistent message.

  - The input message or response might have been put to the dead-letter queue, if one is configured. Look on this queue to see if any new messages are there. This situation is typically accompanied by error messages in the log written by the integration node that describe the problem. For example, the reply-to queue might have been specified incorrectly in the input message, therefore the reply message has been put to the dead-letter queue.

- **Solution:** If your application is not asking for responses (that is, not using messages of type MQMT_REQUEST) consider doing so, particularly when developing applications.

Your application is not receiving publications

Procedure

- **Scenario:** Your application is not receiving publications through the MQ pub/sub broker.
- **Explanation:** If an application has subscribed successfully, it receives publications that match its subscription.

  Subscribers are sent messages only if they match the topic, the subscription point, *and* the filter. Because the subscription point is specified in the message flow, not in the publication message, an incorrect message flow setting can cause unexpected failures.

  A user trace of the flow that contains the Publication node shows you whether publications are matching anything.

- **Solution:** If a filter is being used, a user trace shows you whether this message is being evaluated as expected.

  The case with multiple integration servers, or multiple integration nodes, is more complex. A response is sent to a subscriber after the message has been processed by the target integration server. Other integration servers (and integration nodes) are updated asynchronously. As a result, there might be a delay before publications made elsewhere are received. If the integration node is busy, there can be a delay before messages are processed fully. In a multi-integration node setup, if communications have been suspended, subscription changes are propagated through the network of integration nodes. Check the channels.

  With multiple integration servers or integration nodes, it might be possible to fill intermediate IBM MQ queues if the load is high. This situation might be reported in the syslog (if an integration node cannot put to a queue because it is full) or in the IBM MQ log (if a message coming across a channel cannot be put to the target queue because it is full). If you see messages of this type, display the queue depths on all your queue managers to see if any are almost full.

Publishing a message causes a filter error

Procedure

- **Scenario:** When you publish a message, you receive an error response message with reason text MQRCCF_FILTER_ERROR.
- **Explanation:** An integration node returns this message to a publication when subscriptions have been registered that specify filter expressions (for Content Based Routing) and an error has been encountered when the integration node attempts to filter the published message.
  This situation can occur, for example, if a message is published that includes unsupported data types, or if the message body is corrupted.

Symbols in subscription filters cause problems

Procedure

- **Scenario:** If you specify certain symbols when you use filters in a subscription, the filter does not work.

  Sometimes your subscription messages are put to the dead-letter queue, and a number of error messages are written to the local error log indicating MQRFH2 parsing errors.
- **Explanation:** The MQRFH2 header employs standard XML encoding, so that its parser interprets some symbols in a special way.
- **Solution:** If you want to include these symbols in your filters, use the appropriate escape character to ensure that they are parsed correctly:

Symbol	Escape character
<	&lt;
>	&gt;
"	&quot;
'	&apos;
&	&amp;

  For example, if you want to use:

```
<Filter>Body.e_ALERT_BODY.eqnum<6</Filter>
```

  specify:

```
<Filter>Body.e_ALERT_BODY.eqnum&amp;lt;6</Filter>
```

The Publication node fails with MQRC 2035

Procedure

- **Scenario:** The Publication node fails with MQRC 2035.
- **Explanation:** IBM App Connect Enterprise publishes messages with the user ID in the original message, not the integration node service ID.
- **Solution:** You can force IBM App Connect Enterprise to use the integration node service ID in all circumstances by setting the environment variable *MQSI_PUBSUB_USE_BROKER_USERID* to any value. If there is no MQMD header, or if there is an MQMD header but its `UserIdentifier` field is blank, IBM App Connect Enterprise continues to use the integration node's user ID.

There are performance problems on AIX when the JIT compiler is not loaded

Procedure

- **Scenario:** There are performance problems on AIX when the JIT compiler is not loaded.
- **Explanation:** The environment variable LIBPATH can affect the loading of the Java JIT (just-in-time) compiler on AIX.

  IBM App Connect Enterprise for AIX runs correctly without the JIT compiler, but publish/subscribe performance is adversely affected.

- **Solution:** Ensure that the JIT compiler runs.

  The JIT compiler loads and runs correctly if LIBPATH is not set, therefore do not set LIBPATH. You can make libraries available by linking them into `/var/wmqi/lib` (for all IBM App Connect Enterprise for AIX processes) or `/usr/lib` (for all processes on the system). AIX IBM App Connect Enterprise for AIX configuration does this action for DB2 libraries.

  If it is necessary to set LIBPATH, update it to include the directory `/usr/java130/bin`.

  For example, you can use the following command to start the integration node:

  ```
  LIBPATH=/usr/local/lib:/usr/java130/bin mqsistart mybroker
  ```

Resolving problems with performance

Use the advice given here to help you to resolve common problems with performance.

About this task

- **Scenario:** You are experiencing problems with performance, such as:
  - Poor response times in the IBM App Connect Enterprise Toolkit when developing message flows
  - Poor response time at deployment
  - Individual messages taking a long time to process
  - Poor overall performance, or performance that does not scale well
- **Solution:** Possible solutions are:
  - Tune the integration node
  - Speed up IBM MQ persistent messaging by optimizing the I/O (input/output)
  - Speed up database access by optimizing I/O
  - Increase system memory
  - Use additional instances or multiple integration servers
  - Optimize ESQL statements for best performance

A good source of information about performance is the set of reports in IBM MQ Family Category 2 (freeware) SupportPacs, available for download from the IBM MQ SupportPacs web page.

For more information, read "Do you have a component that is running slowly?" on page 2887.

This topic contains advice for dealing with some common performance problems that can arise:

A WHILE loop in a large XML array takes a long time to process

Procedure

- **Scenario:** A WHILE loop in a large XML array takes a long time to process.

- **Explanation:** This problem arises when you use the CARDINALITY() function to determine the size of the array in the WHILE statement. With large arrays, the cost of determining the number of elements is proportional to the size of the array.

  The CARDINALITY function is invoked each time the WHILE statement is executed. The message has many elements, therefore takes a long time to process when running the loop in this way.
- **Solution:** Unless you have a message in which the number of elements of the array grows dynamically, determine the size of the array before entering the WHILE loop. Use code like the following example:

```
DECLARE i,c INTEGER;
SET i = 1;
SET c=CARDINALITY(OutputRoot.XMLNS.MyMessage.Array[ ]);
WHILE i <= c DO
      . . .
      . . .  loop processing
      . . .
END WHILE;
```

Message flow performance is reduced when you access message trees with many repeating records

Procedure

- **Scenario:** Message flow performance is reduced when the following conditions are true:
  - You are using ESQL processing to manipulate a large message tree.
  - The message tree consists of repeating records or many fields.
  - You have used explicit SET statements with field reference paths to access or create the fields.
  - You have observed a gradual slowing of message flow processing as the ESQL processes more fields or repetitions.
- **Explanation:** This problem occurs when you use field references, rather than reference variables, to access or create consecutive fields or records.

  Consider the following example, in which independent SET statements use field reference paths to manipulate the message tree. The SET statement takes a source and target parameter, where either or both parameters are field references:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field = '1';
```

  The problem arises when the SET statement is used to create many more fields, as shown in the following example:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field2 = '2';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field3 = '3';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field4 = '4';
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field5 = '5';
```

  In this example, the five fields that are created are all children of ParentA. Before the specified field can be created or modified, the integration node must navigate the named message tree to locate the point in the message tree that is to be altered. For example:

  - To access field 1, the SET statement navigates to ParentA, then to the first field, therefore involving two navigations.
  - To access field 5, the SET statement navigates to ParentA, then traverses each of the previous fields until it reaches field 5, therefore involving six navigations.

  Navigating over all the fields that precede the specified field causes the loss in performance.

  Now consider a scenario that accesses repeating fields in an input message tree; for example:

```
DECLARE myChar CHAR;
DECLARE thisRecord INT 0;
```

```
WHILE thisRecord < 10000 DO
 SET thisRecord = thisRecord + 1;
 SET myChar = InputRoot.MRM.myParent.myRepeatingRecord[thisRecord];
END WHILE;
```

When index notation is used, as the count increases, the processing needs to navigate over all the preceding fields to get the one it wants; that is, it has to count over the previous records to get to the one that is represented by the current indexed reference.

- When accessing InputRoot.MRM.myParent.myRepeatingRecord[1], one navigation takes place to get to the first record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[2], two navigations take place to get to the second record.
- When accessing InputRoot.MRM.myParent.myRepeatingRecord[N], *N* navigations take place to get to the N-th record.

Therefore, the total number of navigations for this WHILE loop is: 1 + 2 + 3 + ... + *N*, which is not linear.

- **Solution:** If you are accessing or creating consecutive fields or records, use reference variables. When you use reference variables, the statement navigates to the main parent, which maintains a pointer to the field in the message tree. The following example shows the ESQL that can be used to reduce the number of navigations when creating new output message tree fields:

```
SET OutputRoot.XMLNS.TestCase.StructureA.ParentA.field1 = '1';
DECLARE outRef REFERENCE TO OutputRoot.XMLNS.TestCase.StructureA.ParentA;
SET outRef.field2 = '2';
SET outRef.field3 = '3';
SET outRef.field4 = '4';
SET outRef.field5 = '5';
```

When referencing repeating input message tree fields, you could use the following ESQL:

```
DECLARE myChar CHAR;
DECLARE inputRef REFERENCE TO InputRoot.MRM.myParent.myRepeatingRecord[1];
WHILE LASTMOVE(inputRef) DO
 SET myChar = inputRef;
 MOVE inputRef NEXTSIBLING NAME 'myRepeatingRecord';
END WHILE;
```

For further information, see “Creating dynamic field references” on page 1639.

You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large projects

Procedure

- **Scenario:** You are experiencing poor performance in the IBM App Connect Enterprise Toolkit when working with large or complex projects.
- **Explanation:** Performance is reduced because of frequent project changes, such as adding and removing projects, or using **Project** > **Clean**. Complete project updates use large amounts of memory due to the size, number, and connections between files.
- **Solution:** Increase your system memory.

Performance is reduced when you run web services with small message sizes

Procedure

- **Scenario:** You see poor response times and throughput rates when you run web services using HTTP, and send smaller messages sizes (typically less than 32 KB). Throughput rates can fluctuate with message size. IBM App Connect Enterprise running on the AIX platform might be affected.
- **Explanation:** The default configuration of HTTP enables the Nagle algorithm, which seeks to improve the efficiency of Internet Protocol networks by reducing the number of packets sent. It works

by buffering small packets together, creating a smaller number of large packets. By default, the **tcpnodelay** setting on the sockets of the HTTPRequest is `true`. You can disable the Nagle algorithm at either the operating system level (system wide) or through IBM App Connect Enterprise (affecting only the IBM App Connect Enterprise HTTP sockets).

- **Solution:** Stop the integration node and use the following commands to disable the Nagle algorithm:

  HTTPRequest, SOAPRequest, and SCARequest nodes

  ```
  mqsichangeproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -n tcpNoDelay -v true|false -f
  mqsichangeproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -n tcpNoDelaySSL -v true|false -f
  ```

  To determine the value set, take the following steps:

  Report property values

  Start the integration node and use the following commands:

  ```
  mqsireportproperties integrationNodeName -e integrationServerName
        -o ComIbmSocketConnectionManager -r
  mqsireportproperties integrationNodeName -e integrationServerName
        -o HTTPConnector -r
  mqsireportproperties integrationNodeName -e integrationServerName
        -o HTTPSConnector -r
  mqsireportproperties integrationNodeName -b NodeHttpListener
        -o HTTPConnector -r
  mqsireportproperties integrationNodeName -b NodeHttpListener
        -o HTTPSConnector -r
  ```

Resolving problems when you monitor message flows

Follow the advice for dealing with common problems that can arise when you are monitoring IBM App Connect Enterprise flows.

Procedure

- "You are not receiving monitoring events from IBM App Connect Enterprise" on page 3013
- "Statistics options are not visible in the web user interface" on page 3014

You are not receiving monitoring events from IBM App Connect Enterprise

Procedure

- **Scenario:** You are not receiving monitoring events from IBM App Connect Enterprise.
- **Explanation:** You can configure IBM App Connect Enterprise flows to emit event messages. These messages can be used to support transaction monitoring and business monitoring. The events that are published by IBM App Connect Enterprise can be written to a transaction repository, creating an audit trail of the transactions that are processed by an integration node. You can also use IBM Business Monitor to monitor events. To receive monitoring events, you must configure and enable event sources, activate monitoring, and subscribe to the topic for the flow.

- **Solution:** If you are not receiving monitoring events, check that the following resources are configured correctly.
  a) Check that event sources are configured on the flow by using monitoring properties or a monitoring profile. For more information, see "Monitoring properties and monitoring profiles" on page 2786.
  b) Check that event sources are enabled by setting the **-i** parameter on the **mqsichangeflowmonitoring** command. For more information, see "Enabling and disabling event sources" on page 2801.
  c) Check that monitoring is activated for the flow by setting the **-c** parameter on the **mqsichangeflowmonitoring** command. For more information, see "Activating monitoring" on page 2804.
  d) Check that publication of monitoring events is enabled, and the corresponding pub/sub broker is configured and running.
  Monitoring events are in the `BusinessEvents` group. For more information about configuring the pub/sub broker, and reporting if publication is enabled, see "Configuring and subscribing to performance and monitoring events" on page 2807.
  e) Check that you have subscribed to the topic for the flow.
  For example, on an MQ pub/sub broker:
  ```
  $SYS/Broker/integrationNodeName/Monitoring/integrationServerName/flowName
  ```
  and on an MQTT pub/sub broker:
  ```
  IBM/IntegrationBus/integrationNodeName/Monitoring/integrationServerName/flowName
  ```
  f) If you are writing events to a transaction repository, complete the following extra checks.
  – Check that events are configured for your transactions.
  – Check that you have subscribed to the event topic and written events to a repository.
  g) If you are using IBM Business Monitor to monitor events, complete the following extra checks.
  – Check that IBM Business Monitor is installed, and that a monitor server is created and started. For more information, see IBM Business Process Manager product documentation online.
  – Check that the monitoring model application and message-driven bean (MDB) are installed and configured.
- To report monitoring settings, use the **mqsireportflowmonitoring** command.
  For more information, see command.

Statistics options are not visible in the web user interface

Procedure

- **Scenario:** Your integration node is configured for statistics, but you cannot view or use statistics in the web user interface.
- **Explanation:** The pubsub component was configured, and one of the requirements for statistics is unavailable or disabled. By default, the built-in MQTT broker is used for `OperationalEvents`, which includes statistics.
  For example, if the built-in MQTT broker is disabled and an alternative broker is not specified, statistics will be unavailable.

- **Solution:** Check for one of the following scenarios by using the **mqsireportproperties** command with the **-b** parameter set to pubsub, and take the corrective action:
  a) MQTT is disabled for `OperationalEvents`. Enable `OperationalEvents` over MQTT on the pubsub component.
     For more information, see "Configuring the publication of event messages" on page 2807.
  b) MQTT is enabled for `OperationalEvents` and uses the default policy. However, the built-in MQTT broker is not currently running. Start the MQTT broker.
     For more information, see "Configuring the built-in MQTT pub/sub broker" on page 2811.
  c) MQTT is enabled for `OperationalEvents`, but was configured with an invalid policy URL. Either the syntax is invalid, or the specified policy document does not exist. Check that the URL is valid.
  d) MQTT is enabled for `OperationalEvents` and a valid policy URL was specified. The policy document is valid and refers to an MQTT server. However, the MQTT server is not running. Start the MQTT server.

Resolving problems when developing custom integration applications

Use the advice given here to help you to resolve problems that can arise when developing IBM Integration API applications.

About this task

- "Your custom integration application hangs if the integration node is not available" on page 3015
- "You set a property of an object and query its value, but the value has not changed" on page 3015
- "You cannot connect to an integration node when using a .broker file" on page 3016

Your custom integration application hangs if the integration node is not available

Procedure

- **Scenario:** When the integration node is unavailable, the custom integration application hangs.
- **Explanation:** Communication between the IBM Integration API and the integration node is asynchronous, therefore the custom integration application hangs because it is waiting for a message from the integration node.
- **Solution:** Configure the maximum amount of time that the custom integration application waits by using the following method:

```
// Wait for a maximum of 10 seconds
BrokerProxy.setRetryCharacteristics(10000);
```

  The value specified represents the time in milliseconds that the custom integration application will wait for information before throwing the `BrokerProxyPropertyNotInitializedException` exception.

  If you set this timeout value too low, an exception is thrown, even if the integration node is available.

You set a property of an object and query its value, but the value has not changed

Procedure

- **Scenario:** You have set a property of an object, then queried its value; the value has not changed.
- **Explanation:** Methods that change properties of integration node objects are not processed immediately. If you call a property change method on a custom integration application

object, the IBM Integration API sends a message that requests the specified change to the integration node. The integration node processes the request asynchronously, and notifies all `AdministeredObjectListeners` of the affected object when the change has been attempted.

- **Solution:** Methods that change state typically return to the calling program as soon as the request has been put to the queue manager of the integration node, or, following a call to `BrokerProxy.beginUpdates()`, as soon as the request has been added to the current batch. If the property has still not been updated after the action's response to the request has been returned to the application, consult the response message for more details.

You cannot connect to an integration node when using a `.broker` file

Procedure

- **Scenario:** You cannot connect to an integration node when you use a `.broker` file.
- **Explanation:** If your custom integration applications use the `MQPropertyFileBrokerConnectionParameters` class, they can connect to an integration node by using a connection file that has a `.broker` extension. However, this file can be parsed only if an XML parser is available.
- **Solution:** Ensure that a supported parser is available on the CLASSPATH. A supported parser is shipped with IBM App Connect Enterprise.

  Alternatively, your application can use the `IntegrationNodeConnectionParameters` class instead of the `MQPropertyFileBrokerConnectionParameters` class. This class connects to an integration node by specifying the host name, queue manager name, and web administration port of the target integration node directly. This method does not require an XML parser.

Resolving problems when using an MQ Service

Use the following advice to help you to resolve problems that can arise when you use an MQ Service.

About this task

- "SOAP bindings error when you drag a WSDL file onto Message Flow editor" on page 3016
- "SOAP bindings warning in Problems view when an MQ Service generates a WSDL" on page 3016

SOAP bindings error when you drag a WSDL file onto Message Flow editor

Procedure

- **Scenario:** You drag a WSDL file, that was generated when you created an MQ Service, onto the Message Flow editor. A window opens that contains the error, 'There are no imported SOAP bindings in the WSDL file'.
- **Explanation:** IBM App Connect Enterprise Toolkit expects WSDL files to contain SOAP bindings, but the WSDLs that an MQ Service generates do not contain SOAP bindings.
- **Solution:** The WSDL files that the MQ Service generates are valid. Creating a node by using an MQ Service WSDL file is not supported. To create an MQ node, drag the MQ Service, rather than the WSDL file, onto the Message Flow editor.

SOAP bindings warning in Problems view when an MQ Service generates a WSDL

Procedure

- **Scenario:** You create an MQ Service, and that generates a WSDL file. The Problems view displays the warning, 'The Service does not contain any binding that uses a SOAP 1.1 binding'.

- **Explanation:** The validator expects WSDL files to contain SOAP bindings, and deems a WSDL file invalid if it does not contain them. However, not all WSDL files require SOAP bindings. The WSDL files that an MQ Service generates do not contain SOAP bindings, and are valid.
- **Solution:**
  a) Click **Window** > **Preferences**.
  b) In the Preferences window, open **Integration Development** > **WSDL** > **Validation**.
  c) Check the Allow Services that contain non-SOAP bindings check box and click **Apply**.

Resolving problems with user-defined extensions

Advice for dealing with some common problems that can arise when you work with user-defined extensions

About this task

Procedure

You cannot deploy one of your user-defined nodes, despite having a plug-in LIL in the correct directory.

Procedure

- **Scenario:** You cannot deploy one of your user-defined nodes, despite having a plug-in LIL in the correct directory.
- **Explanation:** You have `memset()` the data area to zero and have not initialized the CNI_VFT structure with the initialization constant `{CNI_VFT_DEFAULT}`.
- **Solution:** Initialize by copying a predefined initialization structure over the function table area, as follows:

  ```
  static CNI_VFT virtualFunctionTable = {CNI_VFT_DEFAULT};
  ```

  In addition, implement logging from your user-defined node so that you can see if the plug-in API is producing error codes; the integration node does not log these to its own log, unless you take a service trace.

You cannot deploy a flow with one of your user-defined nodes in it.

Procedure

- **Scenario:** You cannot deploy a flow with one of your user-defined nodes in it.
- **Explanation:** Your LIL file has failed to load.
- **Solution:** Check system log (syslog or Eventviewer) of integration node startup; did you see a BIP2308 message saying a LIL file failed to load? If there are any problems loading a LIL file, a BIP2308 message appears in the system log.

You get problems when nodes try to use the ESQL path interface in the plug-in API

Procedure

- **Scenario:** When you attempt to use the ESQL path interface in the plug-in API, the return value is CCI_PATH_NOT_NAVIGABLE.
- **Explanation:** The plug-in API allows you to specify a path in the form of an ESQL path expression and navigate to that element, returning a handle to it if it exists. It also allows you to create elements along the path to the requested element.

  The navigate path utility function (`cniSqlNavigatePath`) executes the SQLPathExpression created with the `cniSqlCreateReadOnlyPathExpression` or `cniSqlCreateModifiablePathExpression` utility functions, as defined by the `sqlPathExpression` argument.

  If the path is not navigable, the return code is set to CCI_PATH_NOT_NAVIGABLE. This might be returned when embedding a path expression in another path expression. The input `cciMessage*` functions must not be NULL; however, any of the output `cciMessage*` functions can be NULL. If you embed a path expression that can be NULL inside a path expression that cannot be NULL, CCI_PATH_NOT_NAVIGABLE is returned.
- **Solution:** If the return code is set to CCI_PATH_NOT_NAVIGABLE, ensure that if a correlation name is specified in a path, the respective parameter is not NULL.

After migration your custom property editor does not work

Procedure

- **Scenario:** You have migrated to a new version of IBM App Connect Enterprise and your custom property editor no longer works.
- **Explanation:** Custom property editors can use Eclipse or RAD APIs. If any of those APIs are changed in a new version of IBM App Connect Enterprise, your property editor might not work.
- **Solution:** Update your property editor code to comply with the changed API.

Interpreting problems in user-defined extensions

Procedure

- **Scenario:** You want to debug problems in user-defined nodes and parsers.

- **Solution:** Start user trace at debug level. In order to see BIP4142, BIP4144, BIP4145, and BIP4146 messages, this must be done at the integration server level. For example, use the **mqsichangetrace** command without the **-f** parameter.

  The following debug messages are available to help you to understand the execution of your user-defined nodes and parsers:

  – BIP2233 and BIP2234: a pair of messages that are traced before and after a user-defined extension implementation function is invoked. These messages report the input parameters and returned value. For example:

    BIP2233 `Invoking user-defined extension function [function name] ([function call parameters])`

    BIP2234 `Returned from user-defined extension function [function name] with result: [result of call]`

    Note: In these messages, an *implementation function* can be interpreted as either a C implementation function or a Java implementation method.

  – BIP2308: a message that is logged when the integration node fails to load a LIL file.

    BIP2308 `File [name of LIL file] could not be loaded; operating system return code [error code return from operating system]`

  – BIP3904: a message that is traced before invoking the Java `evaluate()` method of a user-defined node. For example:

    BIP3904 `(for Java): Invoking the evaluate() method of node (class=[node class name], name=[label of node in flow])` where *node class name* is the name of the Java user-defined extension class.

  – BIP3905: a message that is traced before invoking the C `cniEvaluate` implementation function (iFpEvaluate member of CNI_VFT) of a user-defined node. For example:

    BIP3905 `(for C): Invoking the cniEvaluate() implementation function of node (class=[node class name], name=[label of node in flow])` where *node class name* is the name of the user-defined extension class that is provided by the user-defined extension while calling C cniDefineNodeClass.

  – BIP4142: a debug message that is traced when invoking a user-defined node utility function, where the utility function alters the state of a syntax element. This includes all utility functions that start with `cniSetElement*`, where * represents all nodes with that stem. For example:

    BIP4142 `Evaluating cniSetElement [element identifier type]. Changing value from [value before user's change] to [value after user's change]"`

  – BIP4144 and BIP4145: a pair of messages that are traced by certain implementation functions that, when invoked by a user-defined extension, can modify the internal state of an integration node object. Possible integration node objects include syntax element, node, and parser. These messages report the input parameter provided to the invoked method and the returned value. For example:

    BIP4144 `Entered function [function name] ([function call parameters])`

    BIP4145 `Exiting function [function name] with result: [result to be returned]`

    In these messages, an *implementation function* can be interpreted as either a C implementation function or a Java implementation method.

    The C implementation functions that invoke messages BIP4144 and BIP4145 include:

For user-defined parsers	For user-defined nodes
cpiCreateParserFactory	cniCreateElement*
cpiDefineParserClass	cniDeleteMessage
cpiAppendToBuffer	cniAdd*

For user-defined parsers	For user-defined nodes
cpiCreateElement	cniDetach
cpiCreateAndInitializeElement	cniCopyElementTree
cpiAddBefore	cniFinalize
cpiAddAfter	cniWriteBuffer
cpiAddAsFirstChild	cniSql*
cpiAddAsLastChild	cniSetInputBuffer
cpiSetNameFromBuffer	cniDispatchThread

(* represents all nodes with that stem; for example, cniAdd* includes cniAddAfter, cniAddasFirstChild, cniAddasLastChild, and cniAddBefore.)

The Java methods that invoke messages BIP4144 and BIP4145 are:

For user-defined nodes
com.ibm.broker.plugin.MbElement.CreateElement*
com.ibm.broker.plugin.MbElement.add*
com.ibm.broker.plugin.MbElement.detach

For user-defined nodes
com.ibm.broker.plugin.MbElement.copyElementTree

- BIP4146: a debug message that is traced when invoking a user-defined parser utility function, where the utility function alters the state of a syntax element. This includes all utility functions that start with `cpiSetElement*`, where * represents all nodes with that stem. For example:

  ```
  BIP4146 Evaluating cpiSetElement [element identifier type]. Changing value
  from [value before user's change] to [value after user's change]
  ```

  For information on the C user-defined API, see the C language user-defined parser API and the C language user-defined node API.
- BIP4147: an error message that is traced when a user-defined extension passes an invalid input object to a user-defined extension utility API function. For Example:

  ```
  BIP4147 User-defined extension input parameter failed debug validation
  check. Input parameter [parameter name] passed into function [function
  name] is not a valid object.
  ```
- BIP4148: an error message that is traced when a user-defined extension damages an integration node object. For Example:

  ```
  BIP4148 User-defined extension damaged an integration node object. Function
  [function name] has damaged integration node object passed as parameter
  [parameter name].
  ```
- BIP4149: an error message that is traced when a user-defined extension passes an invalid input data pointer to a user-defined extension utility API function. For Example:

  ```
  BIP4149 User-defined extension input parameter failed debug validation
  check. Input parameter [parameter name] passed into function [function
  name] is a NULL pointer.
  ```
- BIP4150: an error message that is traced when a user-defined extension passes invalid input data to a user-defined extension utility API function. For example:

  ```
  BIP4150 User-defined extension input parameter failed debug validation
  check. Input parameter [parameter name] passed into function [function
  name] does not have a valid value.
  ```
- BIP4151: a debug message that is traced when `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to an unexpected value. Expected values are CCI_SUCCESS, CCI_ATTRIBUTE_UNKNOWN, and CCI_BUFFER_TOO_SMALL. Any other value is an unexpected value. For example:

  ```
  BIP4151 An unexpected value was returned from User-defined extension
  implementation function [function name].
  ```
- BIP4152: a debug message that is traced when `cniGetAttribute2` or `cniGetAttributeName2` sets the return code to CCI_BUFFER_TOO_SMALL, and then `cniGetAttribute2` or `cniGetAttributeName2` is called again, this time with the correct size buffer, however the return code is still set to CCI_BUFFER_TOO_SMALL. For example:

  ```
  BIP4152 User-defined extension Implementation function [function name]
  returned CCI_BUFFER_TOO_SMALL on 2nd attempt.
  ```

You want to debug classloading

Procedure

- **Scenario:** You want to debug classloading.
- **Solution:** Classes and the location from which they are loaded are written to user trace. Use this information to check that the correct classes are being loaded.

An error is issued when you deploy a user-defined extension on Linux

Procedure

- **Scenario:** When you deploy a user-defined extension on Linux , an error is displayed in the log of each integration server, stating that there is insufficient authority to open the LIL file.
- **Explanation:** On Linux , the user-defined extension must have group read permission.
- **Solution:**

  On Linux , set the file permissions of the user-defined extension to group read by issuing the command **chmod a+r**

You cannot determine which user-defined extensions have been loaded by the integration node on startup

Procedure

- **Scenario:** You cannot determine which user-defined extensions have been loaded by the integration node on startup.
- **Solution:** Use the **mqsireportproperties** command for each type of user-defined extension.
  - For a Java user-defined extension, issue the command:

```
mqsireportproperties INODE -e default -o ComIbmJavaPluginNodeFactory -r
```

    You see a report similar to this example:

```
ComIbmJavaPluginNodeFactory
 uuid='ComIbmJavaPluginNodeFactory'
 userTraceLevel='none'
 traceLevel='none'
 userTraceFilter='none'
 traceFilter='none'
 NodeClassName='ComIbmJMSClientInputNode'
 NodeClassName='ComIbmJMSClientOutputNode'
 NodeClassName='ComIbmJavaComputeNode'
 NodeClassName='ComIbmXslMqsiNode'
 NodeClassName='SearchFilterNode'

BIP8071I: Successful command completion.
```

    The user-defined extension called SearchFilter has a NodeClassName of SearchFilterNode.
  - For a C user-defined extension (assuming that CONST_PLUGIN_NODE_FACTORY was set to ComIbmSamplePluginNodeFactory in the `NodeFactory.h` file, as in the sample NumComputeNode), issue the command:

```
mqsireportproperties INODE -e default -o ComIbmSamplePluginNodeFactory -r
```

    You see a report similar to this example:

```
ComIbmSamplePluginNodeFactory
 uuid='ComIbmSamplePluginNodeFactory'
 userTraceLevel='none'
 traceLevel='none'
 userTraceFilter='none'
 traceFilter='none'
 NodeClassName='NumComputeNode'

BIP8071I: Successful command completion.
```

    The user-defined extension called NumCompute has a NodeClassName of NumComputeNode.

You are migrating a C user-defined node and cniDefineNodeClass returns CCI_INV_IMPL_FUNCTION.

Procedure

- **Scenario:** When you attempt to migrate a C user-defined node, `cniDefineNodeClass` returns CCI_INV_IMPL_FUNCTION.
- **Explanation:** New fields have been added to the CNI_VFT struct. CNI_VFT_DEFAULT has been updated to initialize these new fields in the header file BipCci.h. If you initialize your CNI_VFT with CNI_VFT_DEFAULT, you should not need to make any code changes. However, if you do not initialize CNI_VFT with CNI_VFT_DEFAULT, these new fields are initialized with random values.
- **Solution:** Initialize your CNI_VFT with CNI_VFT_DEFAULT.

Using logs

There are a variety of logs that you can use to help with problem determination and troubleshooting.

About this task

This section describes how to view the various logs available to you with IBM App Connect Enterprise, and how to interpret the information in those logs. It contains the following topic areas:

Local error log

- "Windows: Viewing the local error log" on page 3023
- "Linux and UNIX systems: Configuring the syslog daemon" on page 3024

Eclipse log

- "Viewing the Eclipse error log" on page 3025

Exception log

- "Viewing the exception log" on page 3026

Activity log

- "Viewing Activity Logs for message flows in the web user interface" on page 2856

Admin log

- "Viewing administration activity in the admin log" on page 330

There is also a section of reference topics about the various types of log.

Windows: Viewing the local error log

The Windows Event Viewer is where IBM App Connect Enterprise writes records to the local system. Use Windows system facilities to view this log.

Viewing the system log

About this task

The system log contains events logged by the Windows system components. For example, the failure of a driver or other system component to load during startup is recorded in the system log. To view the system log:

Procedure

1. Open a command prompt.

2. At the prompt, type **eventvwr**. This opens the Windows Event Viewer.

Viewing the application log

About this task

The application log contains events that are logged by applications or programs. For example, a database program might record a file error in the application log. To view the application log:

Procedure

1. Open a command prompt.
2. At the prompt, type **eventvwr**. This opens the Windows Event Viewer.

Interpreting log information

About this task

In both logs, each event is displayed on a separate row, in date and time order (most recent first), with the following information:

- **Type:** The event type, which can be information, a warning, or an error.
- **Date and time:** The date and time when the event was written to the log.
- **Source:** What action has caused the event.
- **Category:** The category of the event. The default category is none.
- **Event:** The event number.
- **User:** The name of the user at the time of the event.
- **Computer:** The name of the local machine.

To view an individual log entry:

Procedure

1. Within the system or application log, find the log entry.
2. Right-click the entry.
   - On Windows, click **Events** to open the Event Properties window.

     The window shows a description of the event. Select the Details tab to view bytes or words that were parsed when the record was written to the log.
3. Use the up and down arrows to move through the events of the log.
4. To close the window, click **OK** to return to the system or application log.

Linux and UNIX systems: Configuring the syslog daemon

On Linux and UNIX systems, all IBM App Connect Enterprise messages (other than messages that are generated by the command-line utilities) are sent to the syslog subsystem.

About this task

UNIX You must configure this subsystem so that all diagnostic messages that enable you to monitor the performance and behavior of your integration node environment are displayed.

The configuration steps you make to ensure that all relevant messages are displayed depend on the version of Linux and UNIX that you are using. Refer to your operating system documentation relating

to syslog (or syslog-ng for some versions of Linux) for information about how to configure the syslog subsystem.

IBM App Connect Enterprise processes call the syslog commands on the operating system but only those messages that correspond to the filter defined for the output destination are displayed. IBM App Connect Enterprise messages have:

- A facility of `user`.
- A level of `err`, `warn`, or `info`, depending on the severity of the situation causing the message to be issued.

To record all IBM App Connect Enterprise messages, create a filter on the `user` facility for messages of level `info` or greater. It is good practice to write these messages to a separate file; there might be a high number of them and they are more likely to be of interest to broker administrators rather than to system administrators.

The following line in a `syslog.conf` file causes all IBM App Connect Enterprise events to be written to a file `/var/log/user.log`

```
user.info /var/log/user.log
```

Note: Ensure that your log file permissions are set to `-rw-rw-rw`. For example:

```
chmod 666 /var/log/user.log
```

Many UNIX systems provide a command-line utility, known as logger, to help you test and refine your configuration of the syslog subsystem.

On UNIX, syslog entries are restricted in length and messages that are sent to the syslog are truncated by the new line character. To record a large amount of data in a log on UNIX, set the `Destination` property on the Trace node to `File` or `User Trace` instead of `Local Error Log`.

What to do next

See the documentation for your operating system.

Viewing the Eclipse error log

The Eclipse error log captures internal errors that are caused by the operating system or your code.

About this task

To view the Eclipse error log:

Procedure

1. Switch to the Plug-in Development perspective.
2. From the main menu, select **Window** > **Show view** > **Other**. Then select **General** > **Error Log**.

   The error log is displayed, showing the following information for each error:

   - The status of the error (for example, error or warning)
   - A brief description of the error
   - From which plug-in the error derived
   - The date and time that the error was produced
3. If an error has a plus sign (+) at the start of it, it is a complex problem, and there are a number of errors contributing to it. Click the plus sign to view the individual errors.
4. To see the details of a particular problem, double-click the entry in the Problems view.

   A separate window is displayed, showing more details of the error.

Viewing the exception log

The exception log enables the diagnosis of exceptions that occur at any time during processing on an integration server.

About this task

The exception log is controlled by three properties in the `ExceptionLog` subsection of the `ResourceManagers` section of the `server.conf.yaml` configuration file.

The exception log is written to one of the following locations depending on whether the integration server is independent or is managed by an integration node:

- `WorkDirectory/config/common/log/integration_server.`*`IntegrationServerName`*`.exceptionLog.txt` when the integration server is independent.
- `MQSI-REGISTRY/common/log/`*`IntegrationNodeName`*`.`*`IntegrationServerName`*`.exceptionLog.txt` when the integration server is managed by an integration node.

The exception log is a rolling collection of 4 files; when a file is full, data is added to the next file. When the last file has been filled, data starts to be added to the first file, overwriting the previous data there. This cycle of writing continues, ensuring that the newest data is retained. Restarting the integration server causes the previous log to be moved and a numeric suffix added to it.

The exception log includes:

- A summary or short description of the exception
- A list of the exception's inserts
- Message numbers of any nested exceptions
- Optionally, details recorded by the Flow Thread Reporter. If the exception occurs in a message flow, the Flow Thread Reporter provides data to enable you to identify the location in the message flow at which the exception occurred.

Exceptions are listed in the order of their creation.

Procedure

Access the exception log by navigating to the appropriate location:

- `WorkDirectory/config/common/log/integration_server.`*`IntegrationServerName`*`.exceptionLog.txt` if the integration server is independent.
- `MQSI-REGISTRY/common/log/`*`IntegrationNodeName`*`.`*`IntegrationServerName`*`.exceptionLog.txt` if the integration server is managed by an integration node.

Using trace

You can use different types of trace to help you with problem determination and troubleshooting.

About this task

If you cannot get enough information about a particular problem from the entries that are available in the various logs, the next troubleshooting method to consider is using trace. Trace provides more details about what is happening while code runs. The information that is produced from trace is sent to a specified trace record so that you or IBM support personnel can analyze it to discover the cause of your problem.

Trace is inactive by default, and must be explicitly activated by using a command, the IBM App Connect Enterprise web user interface, or the REST API.

There are two main types of trace available in IBM App Connect Enterprise: user trace and service trace. Typically, you run user trace for debugging your applications; you can trace integration nodes, integration servers, and deployed message flows. With service trace, you can activate more comprehensive integration node tracing, and trace the execution of all the commands that are described in commands.

When you start user tracing, you cause extra processing for every activity in the component that you are tracing. Large quantities of data are generated by the components, so you can expect performance to be affected while trace is active. You can limit this extra processing by being selective about what you trace, and by restricting the time during which trace is active.

You can use the following types of trace in IBM App Connect Enterprise:

User trace
You can start, stop, and reset user trace from the IBM App Connect Enterprise web user interface or by using commands. Users of the product in its current form can also investigate the administrative REST API provided by IBM App Connect Enterprise.

For more information, see:

- “User trace” on page 3027
- “Starting user trace” on page 3028
- “Changing user trace options” on page 3029
- “Stopping user trace” on page 3030
- “User trace example” on page 3030

Service trace
Service trace is a type of optional trace that provides more information than that provided by the Event Log or user trace. For more information, see:

- “Service trace” on page 3032
- “Starting service trace” on page 3033
- “Changing service trace options” on page 3035
- “Stopping service trace” on page 3035

For user trace and service trace:

- “Interpreting trace” on page 3036
- “Clearing old information from trace files” on page 3037

ODBC trace
For more information, see “ODBC trace” on page 3037

Integration API trace
For more information, see “IBM Integration API trace” on page 3038

For more information about starting and stopping Trace nodes, see “Switching Trace nodes on and off” on page 3038.

You can also trace a specific command, without modifying trace settings on the integration server, by specifying the trace flag when you run the command. For more information, see the explanation of the **`--trace`** parameter in the relevant command topic.

For more help with data collection, you can use the command command.

User trace

User trace is one of two main types of optional trace that are available in IBM App Connect Enterprise and provides more information than that provided by the entries that are written to the Administration log. User trace is inactive by default; you must activate it explicitly by using a command, the IBM App Connect Enterprise web user interface, or a REST API command.

For information about logs, see Logs. For information about controlling user trace, see "User trace example" on page 3030.

Typically, you use user trace for debugging your applications, as it can trace integration nodes, integration servers, and deployed message flows.

When you activate user trace, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect to see some effect on performance while user trace is active. You can limit this additional processing by restricting the time during which trace is active.

The user trace log files

When trace is active, information is recorded in the user trace log files in plain text format.

For independent integration servers, the location of the trace logs is *workpath*`/config/common/log`, where *workpath* is the work directory of the integration server as specified on the **IntegrationServer** command.

For integration servers that are managed by an integration node, the trace logs are written to `MQSI_WORKPATH/Common/log/`*`integration_node_name`*`.`*`integration_server_name`*`.userTrace.0.txt`

Note: For information about checking and changing the location of the work path; see "Changing the location of the IBM App Connect Enterprise working directory " on page 474.

Trace data is written to files with the format `integration_server.`*`name`*`.userTrace.`*`number`*`.txt` where:

name
The name of your integration server as specified on the **--name** of the **IntegrationServer** command.

number
An integer in the range 0 through 4. There are 5 user trace log files and IBM App Connect Enterprise writes to these files in a rotational manner. When user trace is started, trace data is written to the `integration_server.`*`name`*`.userTrace.0.txt` file. When the `integration_server.`*`name`*`.userTrace.0.txt` file becomes full, user trace data is then written to the `integration_server.`*`name`*`.userTrace.1.txt` file. The process continues until the `integration_server.`*`name`*`.userTrace.4.txt` file is full, at which point user trace data is written to the `integration_server.`*`name`*`.userTrace.0.txt` file and the existing data there is overwritten. The process continues in this rotational manner until user trace is stopped.

Using a Trace node

If you include a Trace node in your message flows when you are developing and testing them, this option not only gives you the ability to trace messages and activity in the flow, but also allows you to specify an alternative target file for the trace contents to isolate the detail in which you are interested. For details of how to use and configure a Trace node, see node and "Switching Trace nodes on and off" on page 3038.

Starting user trace

Use user trace for debugging your applications; you can trace integration nodes, integration servers, and deployed message flows. Start user trace facilities by using the **mqsichangetrace** command, the web user interface, or the REST API.

Before you begin

Before you start to trace an integration node, an integration server, or a deployed message flow, the integration node or server must be running. For more information, follow the instructions in the Chapter 7, "Deploying integration solutions," on page 2463 topic.

About this task

To start a user trace, complete the following steps.

Procedure

1. Start IBM App Connect Enterprise user trace facilities by using the **mqsichangetrace** command, the web user interface, or the REST API:
   - Specify the **-u** parameter on the command.
   - In the IBM App Connect Enterprise web user interface, select **Start user trace** from the options list.
   - Use the REST API provided by IBM App Connect Enterprise.

   You can select only one integration node or server on each invocation of the command, but you can activate concurrent traces for more than one integration node or server, by invoking the command more than once.
2. Specify an individual integration server or message flow within the specified integration node to limit the scope of a trace.

   The events that are recorded when you select the message flow option include:
   - Sending a message from one Message Processing node to the next
   - Evaluating expressions in a Compute or Filter node
3. Start your trace.

   You can start trace at two levels:

   normal
   This tracks events that affect objects that you create and delete, such as nodes.

   debug
   This tracks the beginning and end of a process, as well as monitoring objects that are affected by that process.

Example: starting user trace for an integration server

About this task

To start normal level user trace for integration server `integrationServer1` on integration node `INODE`, enter the command

```
mqsichangetrace INODE -u -e integrationServer1 -l normal
```

where:

-u specifies user trace
-e specifies the integration server (in this case, `integrationServer1`)
-l specifies the level of trace (in this case, normal)

Changing user trace options

Use the **mqsichangetrace** command to change the trace options that you have set.

Procedure

•

Example: changing user trace from normal to debug

About this task

To change from a normal level of user trace to a debug level on the default integration server of an integration node called INODE, on distributed systems, enter the command

```
mqsichangetrace INODE -u -e default -l debug
```

where:

-u specifies user trace
-e specifies the integration server (in this case, the default integration server)
-l specifies the level of trace (in this case, changing it to debug)

Stopping user trace

Use user trace for debugging your applications; you can trace the integration server . Stop user trace facilities by using the **mqsichangetrace** command.

About this task

Use the **mqsichangetrace** command with a trace level of none to stop an active trace. This action stops the trace activity for the component that you specify on the command. It does not affect active traces on other components. For example, if you stop tracing on the integration server `test`, an active trace on another integration server continues.

User trace example

You can start, stop, and reset user trace to control tracing of an integration node or server. Use the **mqsichangetrace** command, the IBM App Connect Enterprise web user interface, or the administrative REST API to control user trace.

Before you begin

Before you start to trace an integration server, the integration server must be running.

About this task

User trace enables you to trace activity in the integration server. User trace is written in plain text format; there is no need to format it before reading it.

To **start** user tracing of the integration server, use one of the following methods:

- In the IBM App Connect Enterprise web user interface, navigate to the top level of the integration server, then select **Start user trace** from the menu at the top-right corner of the page.
- Use the command.
- Use the REST API provided by IBM App Connect Enterprise.

To **stop** user tracing of the integration server, use one of the following methods:

- In the IBM App Connect Enterprise web user interface, navigate to the top level of the integration server, then select **Stop user trace** from the menu at the top-right corner of the page.
- Use the command.
- Use the REST API provided by IBM App Connect Enterprise.

To **reset** user tracing of the integration server, which results in any user trace data that has already been written to the user trace log files being erased, use one of the following methods:

- In the IBM App Connect Enterprise web user interface, navigate to the top level of the integration server, then select **Reset user trace** from the menu at the top-right corner of the page.

- Use the command.
- Use the REST API provided by IBM App Connect Enterprise.

Trace logs:

For independent integration servers, the trace logs are written to *workpath*/config/common/log, where *workpath* is the work directory of the integration server as specified on the **IntegrationServer** command.

For integration servers that are managed by an integration node, the trace logs are written to MQSI_WORKPATH/Common/log/*integration_node_name*.*integration_server_name*.userTrace.0.txt

Example

The following example user trace shows a message that was received by an HTTPInput node being passed to a Mapping node and theMapping node starting to process the data:

```
2018-04-27 09:55:37.845248    12276  UserTrace   BIP3122I: Message received and propagated to
 'out'
terminal of HTTP input node 'Transformation_Map.HTTP Input'.
2018-04-27 09:55:37.845608    12276  UserTrace   BIP3904I: Invoking the evaluate() method of
 node
(class=ComIbmMSLMappingNode, name=FCMComposite_1_3, label=Map).  About to pass a message to the
evaluate() method of  the specified node.  No user action required.
2018-04-27 09:55:37.850016    12276  UserTrace   BIP3964I: The Mapping node 'Map' in message
 flow
'Transformation_Map' is about to process the mapping routine '{default}:Transformation_Map'.
 The Mapping
node is about to process the named mapping routine.  Check the following messages to see the
 progression of
the mapping process.
2018-04-27 09:55:37.851412    12276  UserTrace   BIP11306I: A Parser of type 'WSPROPERTYPARSER'
 has
been created at address '0x248db510'.
2018-04-27 09:55:37.851752    12276  UserTrace   BIP6060I: Node 'Transformation_Map.HTTP Input'
 used
parser type 'Properties' to process a portion of the incoming message of length '0' bytes
 beginning at offset '0'.
2018-04-27 09:55:37.958100    12276  UserTrace   BIP4142I: Evaluating cniElementSetName.
 Changing value
from 'Root' to 'LocalEnvironment'.  Element 'Name' has been changed to 'LocalEnvironment'.  No
 user action
required.
2018-04-27 09:55:37.980888    12276  UserTrace   BIP3963I: The Mapping node is performing a deep
 copy of
the input tree element 'MbElement( type: 1000000 name: Properties )' into the output tree.  The
 Mapping node
is copying the named input tree element, and all its descendents, into the output tree.  Check
 the following
messages to see the progression of the mapping process.
2018-04-27 09:55:37.981508    12276  UserTrace   BIP4145I: Exiting function cniAddBefore with
 result:
CCI_SUCCESS(0).  About to exit the specified the function with the specified result.  No user
 action required.
2018-04-27 09:55:37.984728    12276  UserTrace   BIP3960I: The Mapping node is adding a new
 element with
name 'SaleEnvelopeA' into the output tree.  The Mapping node is adding a new element to the
 output tree, as
defined in the transform that is being processed.  If this element is assigned a value, then
 this assignment will be
reported in an additional message.  Check the following messages to see the progression of the
 mapping process.
2018-04-27 09:55:37.984864    12276  UserTrace   BIP11306I: A Parser of type 'XMLNSC' has been
 created at
address '0x32c2490'.
2018-04-27 09:55:37.984932    12276  UserTrace   BIP4144I: Entered function
cniCreateElementAsLastChild(TRUE, TRUE, N/A, N/A, N/A, N/A, N/A, N/A, N/A, N/A, N/A).  Entered
 the
specified function with the specified parameters.  No user action required.
2018-04-27 09:55:37.984972    12276  UserTrace   BIP4145I: Exiting function
 cniCreateElementAsLastChild
with result: CCI_SUCCESS(0).  About to exit the specified the function with the specified
 result.  No user action
required.
```

Service trace

Service trace is a type of optional trace that is available in IBM App Connect Enterprise, and it provides more information than that provided by the entries that are written to the Event Log or to user trace. Service trace is inactive by default; you must activate it explicitly by using a command.

With service trace, you can activate more comprehensive tracing of an integration node or server. You can also trace the execution of all the commands described in commands, including the trace commands themselves.

Activate service trace only when you receive an error message that instructs you to, or when you are directed to do so by your IBM Support Center. You use the **mqsichangetrace** command to start service trace; you cannot start it through the IBM App Connect Enterprise Toolkit. For more information about how to use service trace, see "Starting service trace" on page 3033.

When you activate service trace, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components, so you can expect performance to be affected while service trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

The location of the trace log depends on whether it is the trace for a command, an integration node (or integration server managed by an integration node), or an independent integration server:

- **Command trace**:

  The location of the trace log for a command is determined by the file name and path that you specify on the **--trace** parameter of the command. When you run a command with trace turned on, you specify the file in which the trace output will be written, as shown in the following example:

  ```
  mqsilist myIntegrationNode --trace c:\tmp\listtrace.txt
  ```

  The trace data is written to the specified file in plain text format (in this example, `c:\tmp\listtrace.txt`). The output is appended to the end of the file; if the file does not exist, it is created automatically.

- **Integration node trace:**

  The location of the trace logs for an integration node (or for an integration server that is managed by an integration node) depends on your environment:

  Windows

  If you have set the work path by using the **-w** parameter of the **mqsicreatebroker** command, the location is `workpath\log`.

  If you have not specified the integration node work path, the default location is `%PROGRAMDATA%\IBM\MQSI\Common\log` where `%PROGRAMDATA%` is the environment variable that defines the system working directory. The default directory depends on the operating system.

  Linux

  `/var/mqsi/common/log`

  When the log file is created, its file name is based on the integration node and integration server to which it relates; for example:

  ```
  myIntegrationNodeName.myIntegrationServerName.trace.n.txt
  ```

  where *n* is a number between 0-3, indicating this specific file in the series of four equally-sized files into which the trace log is divided.

- **Integration server trace:**

  For an independent integration server (which is not managed by an integration node), the trace logs are written to the following location:

  ```
  <workdir>/config/common/log
  ```

  The name of the log file takes the following form:

  ```
  integration_server.myIntegrationServerName.trace.n.txt
  ```

  where *n* is a number between 0-3, indicating this specific file in the series of four equally-sized files into which the trace log is divided.

The directory to which the service trace logs are written must be able to hold all the logs for that computer. You might want to place it on a separate file system, if that is allowed by your system operator.

Starting service trace

Use service trace to get detailed information about your environment for use by your IBM Support Center.

About this task

Activate service trace only when you receive an error message that instructs you to start service trace, or when directed to do so by your IBM Support Center.

Use the **mqsichangetrace** command to start IBM App Connect Enterprise service trace facilities.

You can select only one integration node or server on each invocation of the command, but you can activate concurrent traces for more than one by invoking the command more than once.

To limit the scope of a trace, you must specify the individual integration node or server that you want to trace.

You can trace any IBM App Connect Enterprise command by using the **--trace** parameter. This parameter takes an argument that is the name of the file to which trace records are written. For example, the following command outputs trace of the **mqsistartmsgflow** command to the specified file:

```
mqsistartmsgflow INODE -e eg1 -f simpleflow --trace C:\tracefiles\trace.txt
```

Starting service trace for a command

About this task

To trace an IBM App Connect Enterprise command, use the **--trace** parameter on the command to specify the file to which trace records will be written. For example, the following command outputs a trace of the **mqsilist** command to the specified file:

```
mqsilist myIntegrationNode --trace c:\tmp\listtrace.txt
```

If you want to trace the command executable files themselves, set the environment variables MQSI_UTILITY_TRACE and MQSI_UTILITY_TRACESIZE before you run the command.

MQSI_UTILITY_TRACE
Set this variable to `normal` for a basic level of trace, or `debug` for a fuller trace.

MQSI_UTILITY_TRACESIZE
The size, in kilobytes, of the binary trace file. See the **-c** parameter of the **mqsichangetrace** command.

Ensure that you reset these variables when the command that you are tracing has completed. If you do not do so, all subsequent commands are also traced, and their performance will be degraded.

Starting service trace for an integration node

About this task

To start a debug-level service trace for the integration node INODE, enter the following command:

```
mqsichangetrace INODE -t -b -l debug
```

where:

-t specifies service trace
-b specifies that trace for the `agent` subcomponent of the specified component is to be started
-l specifies the level of trace (in this case, debug)

Starting service trace for an integration server that is managed by an integration node

About this task

To start a debug-level service trace for an integration server IS1 on INODE, enter the following command:

```
mqsichangetrace INODE  -t -e IS1 -l debug -m fast -c 200000 -r
```

where:

-t specifies service trace
-l specifies the level of trace (in this case, debug)
-m specifies the way trace information is to be buffered (in this case, fast)
-c specifies the size of the trace file in KB (in this case, 200000)
-r specifies that the trace file is reset

Starting service trace for an independent integration server

About this task

You can start service trace for an independent integration server, either by setting the **trace** property in the integration server's `server.conf.yaml` file, or by running the **IntegrationServer** command with the **--service-trace** parameter:

- Start service trace for an independent integration server by setting the **trace** property in the integration server's `server.conf.yaml` file, as shown in the following example:

  ```
  trace: 'service'
  ```

  The settings in the `server.conf.yaml` file take effect when the integration server is restarted. For more information about configuring an integration server, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172.
- Start an independent integration server with service trace enabled, by running the following command:

  ```
  IntegrationServer --name myIntegrationServer1 --work-dir c:\mywrk\myaceworkdir --service-trace
  ```

  For more information about starting an integration server, see command.

Changing service trace options

Use the **mqsichangetrace** command to change the service trace options that you have set.

Example: changing service trace from debug to normal

About this task

To change from a debug level of trace to a normal level on the integration node, on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -b -l normal
```

where:

-t specifies service trace
-b specifies that tracing for the `agent` subcomponent of the specified component is to be changed
-l specifies the level of trace (in this case, changing it to normal)

z/OS On z/OS, enter the following command:

```
F MQP1BRK,ct t=yes, b=yes, l=normal
```

Stopping service trace

Use the **mqsichangetrace** command to stop an active service trace.

Procedure

- Use the **mqsichangetrace** command with a trace level of `none` to stop an active trace.

  This action stops the trace activity for the component that you specify on the command. It does not affect active traces on other components. For example, if you stop tracing on the integration server `test`, an active trace on another integration server continues.

Results

If you redeploy a component from the IBM App Connect Enterprise Toolkit, trace for that component is returned to its default setting of `none`.

Example: stopping service trace on an integration server, using the `mqsichangetrace` command

About this task

To stop debug level service tracing for an integration server `IS1` on integration node A on distributed systems, enter the command

mqsichangetrace `BrokerA -t -e IS1 -l none`

where:

-t specifies service trace
-l specifies the level of trace (in this case, none)

z/OS On z/OS, enter the command

```
F MQPIBRK,ct t=yes, b=yes, l=none
```

Example: stopping service trace on the integration node

About this task

To stop the trace started by the command shown in "Starting service trace" on page 3033, on distributed systems, enter the following command:

```
mqsichangetrace BrokerA -t -b -l none
```

where:

-t specifies service trace
-b specifies that trace for the `agent` subcomponent of the specified component is to be stopped
-l specifies the level of trace (in this case, none)

z/OS On z/OS, enter the following command:

```
F MQP1BRK,ct t=yes, b=yes, l=none
```

Interpreting trace

Use the information in a formatted trace file to identify unexpected behavior.

About this task

A formatted log file contains a sequence of IBM App Connect Enterprise messages. These messages record the activity in a specific part of the system (the part that you identify when you start the trace). You can use this sequence to understand what is happening, and to check that the behavior that is recorded is what you are expecting.

For example, message flow trace records the path that a message takes through the message flow. You can see why decisions result in this path (where a choice is available).

Procedure

1. Verify that the trace file is complete.

   If the size of the trace log is too small to contain all events, trace output continues at the start of the trace log, overwriting existing entries. This is known as *wrapping*.

   An indication that a trace has wrapped is the relationship between the first and last timestamp in it, and the time that trace was enabled. For example, assume that you start tracing at 10:15, and collect the trace at 10:30. If the trace timestamps run from 10:20 to 10:30, it is probable that trace wrapped. Of course it could mean that nothing happened between 10:15 and 10:20.

   Examine the trace and decide whether the beginning of it makes sense, and whether it looks complete. For example, if you want to trace the passage of three messages through a flow, and trace starts half way through the second message, it could have wrapped, or trace might not have been enabled early enough.
2. If trace has wrapped, increase the size of the trace file, and rerun trace.

   For information on trace settings, see command.
3. If you see unexpected behavior in a message flow or integration server, use this trace information to check the actions that have been taken and identify the source of an error or other discrepancy.

Results

The messages contain identifiers for the resources that are being traced, for example the integration servers and message flows. The identifier that is given is typically the label (the name) that you gave to the resource when you defined it.

Here is an extract from a user trace file. In the example, each column has been labeled:

```
Timestamp                 Thread ID Trace type  Message
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2537I: Node 'Reply.MapToRequestor':
                                                Executing statement   ''SET I = I + 1;''
                                                at ('.MapToRequestor.CopyMessageHeaders',
                                                '6.4').
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2539I: Node 'Reply.MapToRequestor':
                                                Evaluating expression ''I'' at
                                                ('.MapToRequestor.CopyMessageHeaders',
                                                '6.12'). This resolved to ''I''. The
                                                result was ''1''.
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2539I: Node 'Reply.MapToRequestor':
                                                Evaluating expression ''I + 1'' at
                                                ('.MapToRequestor.CopyMessageHeaders',
                                                '6.14'). This resolved to ''1 + 1''.
                                                The result was ''2''.
2005-07-12 16:17:18.242605   5344   UserTrace   BIP2566I: Node 'Reply.MapToRequestor':
                                                Assigning value ''2'' to field / variable
                                                ''I''.
```

References such as `'6.12'` apply to the row and column number within a function that specify the location of the command that is being executed; in this case, row 6, column 12.

Clearing old information from trace files

If the component that you are tracing has stopped, you can delete its trace files from the log subdirectory of the IBM App Connect Enterprise home directory.

About this task

If you are tracing an integration server, you can use the **-r** parameter of the **mqsichangetrace** command to reset (clear) the trace log (the **-r** parameter can be specified only if you specify the **-e** parameter). You might clear the log when you start a new trace, to ensure that all the records on the log are unique to the new trace.

Example: clearing the user trace log for the default integration server

About this task

To clear the user trace log for the default integration server, on distributed systems, enter the command: **mqsichangetrace WBRK_BROKER -u -e default -r** where:

WBRK_BROKER specifies the name of the integration node
-u specifies user trace
-e specifies the integration server (in this case the default integration server)
-r clears the trace log

ODBC trace

You can use various methods to trace for ODBC activity, depending on the operating system that you are using.

About this task

Windows For Windows, use the **Tracing** tab of the ODBC function:

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools**.
2. Double-click **Data Sources**.
3. Click the **Tracing** tab.
4. Click **Start Tracing Now**.
5. Click **OK**.

To stop ODBC tracing, on the **Tracing** tab, click **Stop Tracing Now**, then **OK**.

Linux UNIX For Linux and UNIX operating systems using IBM Integration ODBC Database Extender drivers:

- To initiate trace for ODBC activity, edit the [ODBC] stanza in the `odbcinst.ini` file in the directory pointed to by your ODBCSYSINI environment variable as follows:
  1. Change `Trace=no` to `Trace=yes`.
  2. Specify a path and file name for `TraceFile`
  3. Ensure that the `TraceFile` entry points to a file system that has enough space to receive the trace output

IBM Integration API trace

Enable or disable service trace for the IBM Integration API.

Enabling IBM Integration API trace

About this task

To enable tracing for the IBM Integration API for your application, set the environment variable MQSI_IAPI_TRACE, where *<filename>* is the name of the file to which the trace is sent:

```
export MQSI_IAPI_TRACE=<filename>
```

Or you can enable tracing by using the following API call in your code:

```
// Enable Administration service trace
     IntegrationAPI.enableAdministrationAPITracing("outputfile.txt");
```

This request logs all calls to the IBM Integration API to the `outputfile.txt` file in the current directory. All IBM Integration API activity in the entire Java virtual machine is logged.

You can also enable IBM Integration API service trace from the **File** menu of the IBM Integration API Exerciser. Alternatively, you can enable IBM Integration API tracing in the IBM App Connect Enterprise Toolkit by clicking **Window** > **Preferences** > **Unit Test Settings** > **Integraton API**.

Disabling IBM Integration API trace

About this task

To disable tracing for the IBM Integration API for your application, use the following API call in your code:

```
// Disable Administration service trace
   IntegrationAPI.disableIntegrationAPITracing()
```

You can also disable IBM Integration API service trace from the **File** menu of the IBM Integration API Exerciser.

Switching Trace nodes on and off

You can enable and disable Trace nodes by setting properties in the `server.conf.yaml` configuration file.

About this task

When an integration server is created, or a message flow is deployed, its Trace node switch is set to on by default. The message flow level Trace node switch setting is not changed on redeployment. You can significantly improve the performance of a flow that includes Trace nodes by switching Trace nodes off.

If the Trace node setting for an integration server is off, all Trace nodes in its flows are disabled. You can change the settings for Trace nodes in individual message flows; the settings are applied when you turn on Trace nodes for the integration server. If the Trace node setting for an integration server is on, the Trace node switch setting of each message flow determines the effective settings.

Example: switching off Trace nodes for an integration server, by using the server.conf.yaml file

Procedure

1. Open the `server.conf.yaml` configuration file for your integration server, by using a YAML editor.

   If you do not have access to a YAML editor, you can edit the file by using a plain text editor; however, you must ensure that you do not include any tab characters, which are invalid in YAML and would cause your integration server configuration to fail. If you are using a plain text editor, ensure that you use a YAML validation tool to validate the content of your file.
2. Set the **`traceNodeLevel`** property to `false`.

   For more information about configuring an integration server by using the `server.conf.yaml` file, see “Configuring an integration server by modifying the server.conf.yaml file” on page 172.
3. Restart the integration server for the changes to take effect.

   For information about how to start an integration server, see “Starting an integration server” on page 250.

Using dumps and abend files

A dump or an abend file might be produced when a problem occurs. Dumps and abend files can be used by IBM to resolve the problem.

About this task

Refer to the topics in the Procedure section:

Procedure

- “Checking for dumps” on page 3039
- “Checking for abend files” on page 3040

What to do next

You can also use the command command to help with data collection.

Checking for dumps

If a dump occurs on your system, an error message is produced.

Procedure

- **On Windows**

  BIP2111 error message (IBM App Connect Enterprise internal error). The error message contains the path to the MiniDump file in your errors directory.
- **On UNIX**

  BIP2060 error message (integration server terminated unexpectedly). Look in the directory where the integration node was started, or in the service user ID's home directory, to find the core dump file.

Checking for abend files

Abend files are produced when a process ends abnormally. The information contained in an abend file helps the IBM Support Center to diagnose and fix the problem.

About this task

The following list contains examples of what might cause the integration node to produce an abend file:

Procedure

- The integration node runs out of memory.
- A user-defined extension causes an instruction in the integration node process that is not valid.
- An unrecoverable error occurs in the integration node.

Results

Abend files are never produced during normal operation. If an abend file is produced, contact the IBM Support Center for assistance.

Contacting your IBM Support Center

If you cannot resolve problems that you find when you use IBM App Connect Enterprise, or if you are directed to do so by an error message generated by IBM App Connect Enterprise, you can contact your IBM Support Center.

About this task

Before you contact your Support Center, use the checklist that is shown here to gather important information. Some items might not be relevant in every situation, but provide as much information as you can to enable the IBM Support Center to re-create your problem. You can also use the command to collect diagnostics.

For IBM App Connect Enterprise:

- The product version.
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or LinuxUNIX operating system syslog entries, and any abend or dump files from the *install_dir*\errors directory on Windows, or the /var/mqsi/errors directory on Linux UNIX. Obtain user trace log files at debug level for all relevant message flows and preferably format them. Also, include any requested service trace files.

  To send files from distributed systems, create a compressed file by using any compression utility.

  To send a file from the file system to IBM, use **tar** to compress the file. For example, `tar -cx -f coredump.0002009E coredump.toibm`. To send MVS data sets to IBM, terse them using TRSMAIN, which you can download from z/OS tools download.
- A list of the components installed. Include details of the number of computers and their operating systems, the number of integration nodes and the computers on which they are running.
- The compressed file that is obtained by exporting your workspace and appropriate message flows and message sets. This task is done in the IBM App Connect Enterprise Toolkit.
- Details of the operation that you were running, the results that occurred, and the results that you were expecting.
- A sample of the messages that were being used when the problem arose.

- If relevant, the report file from the C or COBOL importer. This file is located in the directory from which the file import was attempted.
- If you are using tagged delimited wire format on message sets, include the Tivoli Directory Server log files.

For IBM MQ:

- The product version.
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or Linux and UNIX operating system syslog entries and First Failure Support Technology (FFST) output files. You can find these files, which have the extension `.fdc`, in the errors subdirectory in the IBM MQ home directory.
- Details of IBM MQ client software, if appropriate.

For each database that you are using:

- The product and release level (for example, DB2 9.1).
- Any fix packs applied.
- Any interim fixes applied.
- All current trace and error logs, including relevant Windows Event log or Linux and UNIX operating system syslog entries, for example the `db2dialog.log` file on DB2. Check the database product documentation for details of where to find these files.
- Definitions of any database tables.
- Any ODBC traces.

Windows For Windows:

- The version.
- The Service Pack level.
- The environment settings.

UNIX For Linux and UNIX operating systems:

- The product version. You can find the version that is installed by using the **`uname -a`** command.
- Any service level and patches that are applied.
- The environment settings.

Collecting diagnostics

You can collect diagnostic documents by using the **`aceDataCollector`** command.

About this task

You can run the command that is provided in the following topic to collect the appropriate information for IBM App Connect Enterprise.

Procedure

- command

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Procedure

1. **Search the product documentation**

   IBM provides extensive documentation. Product documentation can be installed on your local computer or on a local intranet. Product documentation can also be viewed on the IBM website. You can use the powerful search function of the product documentation to query conceptual and reference information as well as detailed instructions for completing tasks.

2. **Search the Internet**

   If you cannot find an answer to your question in the product documentation, search the Internet for the latest, most complete information that might help you resolve your problem, including:

   - IBM technotes
   - IBM downloads
   - IBM Redbooks publications
   - IBM developerWorks
   - Forums and newsgroups
   - Internet search engines

Getting product fixes

A product fix might be available to resolve your problem. You can determine what fixes are available from the IBM support site.

About this task

To determine what fixes are available from the IBM support site:

1. Open the IBM App Connect Enterprise support web page.
2. Click **Downloads**, then **Recommended fixes**. This web page provides links to the latest available maintenance for the in service IBM App Connect Enterprise family of products.

To receive weekly email notifications about fixes and other news about IBM products, follow these steps.

Procedure

1. From the support site (IBM App Connect Enterprise support web page), locate the **Notifications** box in the center of the page.
2. If you are not signed in, click **Sign in to create, manage or view your subscriptions**. If you have not registered, click **register now** on the sign-in page and follow the on-screen instructions.
3. Click **Manage my subscriptions**.
4. Click the **Subscribe** tab. A list of products families is shown.
5. In the Software column, click **WebSphere**. A list of products is shown.
6. Select the product for which you want to receive notifications (for example, **IBM Integration Bus**), then click **Continue**.
7. Set options to determine what notifications you receive, how often you receive them, and to which folder they are saved, then click **Submit**.

Contacting IBM Software Support

IBM Software Support assists with product defects.

About this task

Before you contact IBM Software Support, you must ensure that your company has an active IBM software subscription and support contract, and that you are authorized to submit problems to IBM. The type of software subscription and support contract that you need depends on the type of product that you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational products, DB2 and WebSphere products that run on Windows or UNIX operating systems). Enroll in Passport Advantage in one of the following ways:
  - Online: Go to the Passport Advantage website and click **How to Enroll**.
  - By telephone: For the telephone number to call in your country, go to the Software Support Handbook, click **Contacts**, then **Worldwide contacts**.
- For customers with Subscription and Support (S & S) contracts, go to the Open service request web page.
- For customers with IBMLink, CATIA, Linux, S/390®, iSeries, pSeries, zSeries, and other support agreements, go to the IBM Support Line web page.
- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software subscription and support agreement by working directly with an IBM marketing representative or an IBM Business Partner. For more information about support for eServer software products, go to the Support for IBM Systems web page.

If you are not sure what type of software subscription and support contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. From other countries, go to the contacts page of the Software Support Handbook and click the name of your geographic region for telephone numbers of people who support your location.

Complete the following steps to contact IBM Software Support:

Procedure

1. “Determine the effect of the problem on your business” on page 3043
2. “Describe your problem and gather background information” on page 3044
3. “Submit your problem to IBM Software Support” on page 3044

Determine the effect of the problem on your business

About this task

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the effect on your business of the problem that you are reporting. Use the following criteria:

Severity 1
: **Critical** effect on business: You are unable to use the program, resulting in a critical effect on operations. This condition requires an immediate solution.

Severity 2
: **Significant** effect on business: The program is usable but is severely limited.

Severity 3
: **Some** effect on business: The program is usable with less significant features (not critical to operations) unavailable.

Severity 4

Minimal effect on business: The problem has little effect on operations, or a reasonable workaround to the problem was implemented.

Describe your problem and gather background information

About this task

When you are explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you to solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be re-created? If so, what steps led to the failure?
- Have you changed the system? (For example, changes to hardware, operating system, networking software.)
- Are you currently using a workaround for this problem? If so, be prepared to explain it when you report the problem.
- The **aceDataCollector** command can be used to collect diagnostic information. For more information, see command.

Submit your problem to IBM Software Support

About this task

You can submit your problem in the following ways:

- Online: Go to the Software Support Handbook and enter your information into the appropriate problem submission tool.
- By telephone: For the telephone number to call in your country, go to the contacts page of the Software Support Handbook. Click the name of your geographic region for telephone numbers of people who support your location.

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support might create an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered.

IBM publishes resolved APARs on the IBM product support web pages daily so that other users who experience the same problem can benefit from the same resolutions.

Chapter 11. IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX)

IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) enables z/OS customers to run and manage IBM App Connect Enterprise in zCX by using JCL or z/OS console commands.

About this task

You can configure IBM App Connect Enterprise to run on IBM z/OS Container Extensions (zCX) as described in "Configuring App Connect Enterprise to run in Docker" on page 111.

If you prefer to use JCL or IBM z/OS console commands to administer integration servers, and run IBM App Connect Enterprise commands, you can install and configure IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX). IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) provides the following resources:

- JCL. You can use the supplied JCL to build or load Docker images and administer the integration server Docker containers.
- Shell scripts. Shell scripts are used to interact with the zCX instance and issue commands. You run the shell scripts by editing and submitting the supplied JCL or by running the supplied console listener program.
- Console listener program. You can use the console listener program if you want to administer integration server Docker containers by using IBM z/OS console commands.

Procedure

Install and configure IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by completing the steps in the following topics:

- "Provisioning an IBM z/OS Container Extensions (zCX) instance" on page 3046
- "Installing and configuring IBM App Connect Enterprise on zCX" on page 3046
- "Customizing the JCL" on page 3048

You can then create an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by completing the following task:

- "Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL" on page 3049

You can then use either JCL or z/OS console commands to manage your integration servers and to run IBM App Connect Enterprise commands. Complete the steps described in one of the following topics, according to your chosen method:

- "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL" on page 3053
- "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3056

The following topics contain additional reference information, which you can use to help you complete the relevant tasks:

- "Planning and customization tasks on z/OS" on page 3061
- "Environment variables in STDENV and BIPENVS" on page 3062
- "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071
- Troubleshooting on IBM z/OS Container Extensions (zCX)
- Applying maintenance toIBM z/OS Container Extensions (zCX)

Provisioning an IBM z/OS Container Extensions (zCX) instance

Provision an IBM z/OS Container Extensions (zCX) instance.

Procedure

1. If you do not have an available instance of IBM z/OS Container Extensions (zCX), provision an instance as described in z/OS Container Extensions (zCX) content solution.
2. If they do not exist, create one or more Docker users in zCX to run the integration server processing as described in Local user management.
3. For each user in IBM z/OS Container Extensions (zCX), create an ssh private/public key pair to be used to run the integration server processing.
   a. Create the keys by using the `ssh-keygen` command, or create them according to your installation standards. Do not set a passphrase. For example, you can use the following command to generate a private/public key pair:

      ```
      ssh-keygen -t rsa -b 4096 -C "your_email@domain.com"
      ```

      The private key is named *id_rsa*, and the public key is named *id_rsa.pub*, or the keys are named as you specified when you generated them. You can run the command from the z/OS UNIX System Services file system, but you are not obliged to do so. If you run the command from the z/OS UNIX System Services file system, the keys are created by default in the `.ssh` subdirectory of your home directory. For example, `/u/aceadmin/.ssh`. For more information, see Steps for setting up user authentication when using UNIX files to store keys, and Local user management in the IBM z/OS documentation online.
   b. Copy the public ssh key into the IBM z/OS Container Extensions (zCX) instance. For example, if you created the key for the Docker user *zcxuser* in your home directory of the z/OS UNIX System Services file system, run the following command:

      ```
      cat ~/.ssh/id_rsa.pub | ssh -p 8022 zcxuser@9.20.11.10 'cat >> .ssh/authorized_keys &&
       echo "Key copied"'
      ```

   c. Store the private key on the z/OS UNIX System Services file system for use when you run the integration server processing. The private key must be set as the *ZCX_SSH_KEY* environment variable, and the associated Docker user must be set as the *ZCX_USER* environment variable.

   For more information about access options, see Configuring SSH access for IBM z/OS Container Extensions (zCX).

What to do next

Install and configure IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX), as described in “Installing and configuring IBM App Connect Enterprise on zCX” on page 3046.

Installing and configuring IBM App Connect Enterprise on zCX

Install and configure IBM App Connect Enterprise on zCX.

About this task

You can install and configure IBM App Connect Enterprise on zCX by completing the following steps:

Procedure

1. Go to IBM Passport Advantage (https://www.ibm.com/software/passportadvantage/pao_customer.html) and download the appropriate IBM App Connect Enterprise Linux on Z for IBM zCX Multilingual portable software instance image.

2. Transfer the image to your z/OS UNIX file system. If you are using FTP to transfer the file, ensure that the mode is set to `binary`.
3. Extract the files from the downloaded image, by using the `pax -rvf image.pax.Z` command. This command extracts the files into the current directory.
4. Log on to the IBM z/OS Management Facility for your z/OS system.
5. In the main navigation, expand **Software** and select **Software Management**.
6. In the **Software Management table**, click **Portable Software Instances**.
7. In the table header, click the **Actions** drop-down menu and select **Add from z/OSMF system**.
8. Select the following details for where your image is stored:
   - **System**. The logical partition (LPAR) where your image is located. Select this value by using the drop-down list or the **Select** button.
   - **File location**. The file system location for the image; for example `/u/jo/ace`.
9. Click **Retrieve**, check that the IBM App Connect Enterprise PSI is shown in the table, then click **OK**.
10. In the table, select **IBM App Connect Enterprise**. In the **Switch To** drop-down menu, select **Deployments**.
11. Initially, no deployments are displayed. In the table header, click the **Actions** drop-down menu and select **New** to create a new one.
12. In the **Deployment Checklist**, click **Specify the properties for this deployment**.
13. Enter a name for the deployment; for example, *ACE*. Click **OK**.
14. In the **Deployment Checklist**, click **Select the software to deploy**.
15. Set the software type by clicking **Portable Software Instance**.
16. Select the PSI that you created previously, then click **OK**.
17. In the **Deployment Checklist**, click **Select the objective for this deployment**.
18. In the **Select Deployment Objective** tab, select the following parameters:
    - Set **Objective** to **Create a new software instance and connect to the following global CSI**, then select **A new global zone CSI**.
    - Set **Target system** to be the LPAR to install on.
19. Click **OK**.
20. In the **Deployment Checklist**, click **Check for missing SYSMODs**.
21. Click **Next**, then clear the following reports, which are not required for a new installation:
    - Requisite SYSMODs and Fix Categories reports.
    - Regressed SYSMODs and HOLDDAATA Delta reports.
22. Click **Finish** and ignore the warning that is issued when no reports are selected.
23. In the **Deployment Checklist**, click **Configure this deployment**.
24. Work through the configuration wizard. To use the existing values, click **Next**, or to change the values, click **Modify**.
25. When the wizard is complete, in the **Deployment Checklist**, click **Define the job settings. z/OSMF creates the deployment summary and jobs.**
26. Make updates to the **JOB statement** as required. When the changes are complete, click **OK**.
27. In the **Deployment Checklist**, click **Submit deployment jobs**.
28. The jobs in the **Submit Deployment Jobs** tab must be submitted one at a time in the specified sequence. When a job completes successfully, submit the next job. To submit a job, select the checkbox next to the job, then open the **Actions** drop-down menu and click **Submit**.
29. When all the jobs are completed, the file system is ready, but is not mounted. Update the SYS1.PARMILIB BPXPRMxx configuration file to mount the file system. The standard mount point for IBM App Connect Enterprise is `/usr/lpp/ace/v12/`.
30. In the **Deployment Checklist**, click **Specify the properties for the target software instance**.

31. By default, the software instance name and description are copied from the source software instance, but you can edit them. When the name and description are appropriate, click **OK**. Software deployment is complete.
32. Ensure that the environmental requirements for IBM z/OS Container Extensions (zCX) are met, as described in "Planning and customization tasks on z/OS" on page 3061.

Results

The IBM z/OS Management Facility installation process on zCX provides the following libraries and file systems:

Table 103. Libraries and file systems that are provided by the IBM z/OS Management Facility installation process:

Name	Description
`<hlq>.SACECNTL`	Contains sample JCL
`<hlq>.SACELOAD`	Contains the load module for the console listener task.
Mounted file system	Contains the shell scripts and sample Dockerfile to build the docker image.

What to do next

Customize the JCL as described in "Customizing the JCL" on page 3048.

Customizing the JCL

Customize the JCL, as part of creating an integration server on IBM z/OS Container Extensions (zCX).

About this task

All JCL has a standard header, which includes the following items:

- A brief description of its function.
- A description where further information can be found.
- Instructions about how to customize the JCL.
- A section that lists the JCL itself.

The following sample JCL is supplied by the IBM z/OS Management Facility installation process:

- `BIPXBLD` - Sample job to build a Docker image to run an integration server.
- `BIPXCLIS` - Sample job to run the console listener program.
- `BIPXDBG` - Sample job to debug the integration server Docker container.
- `BIPXDLI` - Sample job to load a Docker image from a `.tar` archive on z/OS UNIX System Services.
- `BIPXDSP` - Sample job to run a Docker system purge to clear space on the zCX instance.
- `BIPXGET` - Sample job to copy or move a file from a running integration server Docker container to UNIX System Services.
- `BIPXIS` - Sample job to run an integration server in zCX and display its logs.
- `BIPXISCM` - Sample job to run an integration server runtime command.
- `BIPXISTP` - Sample job to stop the integration server or to stop and remove its container, or to remove the container that is already stopped.
- `BIPXPUT` - Sample job to copy a file from UNIX System Services to a running integration server Docker container.

Procedure

1. Customize the JCL files by following the instructions in the files. You can customize the files by using an ISPF edit macro that you must tailor, or you can edit each of the PDSE members manually. Most of the customizations are standard for all of the JCL files.
2. Create environment files and set the environment variables that you need for each task.

   You need a different set of environment variables for each task such as building an image, starting an integration server, or running commands. Therefore, you can choose to create a different environment file for each task and or integration server. For each task, you might need more than one environment file if different users have different levels of access. You can specify multiple environment files in the *STDENV* concatenation. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.

What to do next

Create an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX), as described in "Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL" on page 3049.

Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL

Create an IBM App Connect Enterprise integration server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL.

About this task

You can build your own integration server Docker image by using the supplied JCL. The image must be made available to the zCX instance. If you choose not to build your own image by using the supplied JCL, you can use any image that is built for the s390x architecture. The following alternatives are available:

- You can use an existing image in any Docker registry to which your zCX instance is connected. For more information, see Docker registries in the IBM z/OS documentation online.
- You can load a docker image from a `.tar` archive in the z/OS UNIX System Services file system. For more information, see "Loading a Docker image from a .tar file on the z/OS UNIX System Services file system" on page 3051.
- You can build your own Docker image by using your own processing methods as described in "Building your own Docker image" on page 3052.

Procedure

To build your own integration server Docker image by using the supplied JCL, complete the following steps:

1. Download an IBM App Connect Enterprise Linux on IBM Z package from Passport Advantage / PassportAdvantage Express Overview and place it in the UNIX System Services file system on z/OS.

   Do not to use Secure Copy Protocol (SCP) to transfer the package. SCP assumes that files are text, so binary files that are copied between EBCDIC and ASCII operating systems are corrupted during the copying process.

2. Set the environment variables in the environment file, ENVFILE, as described in "Environment variables in STDENV and BIPENVS" on page 3062.

   You must set the zCX access environment variables and the environment variables that are specific to creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX).

   For more information about access options, see Configuring SSH access for IBM z/OS Container Extensions (zCX).

   - Specify the location of the downloaded Linux on IBM Z package. For example:

     ```
     ACE_DOWNLOAD_PACKAGE_DIR=/u/aceadmin
     ```

   - Specify the name of the downloaded Linux on IBM Z package. For example:

     ```
     ACE_DOWNLOAD_PACKAGE_NAME=ace-12.0.1.0.tar.gz
     ```

   - If you want to include a BAR file, place it in the UNIX System Services file system on z/OS and set the environment variables *ACE_DOCKER_IMAGE_BAR_DIR*, and *ACE_DOCKER_IMAGE_BAR_NAME*. For example:

     ```
     ACE_DOCKER_IMAGE_BAR_DIR=/u/aceadmin/bars
     ```

     ```
     ACE_DOCKER_IMAGE_BAR_NAME=zcx.bar
     ```

   - If you want to include a `server.conf.yaml` file, place it in the UNIX System Services file system on z/OS and set the environment variables *ACE_DOCKER_IMAGE_YAML_DIR*, and *ACE_DOCKER_IMAGE_YAML_NAME*. For example:

     ```
     ACE_DOCKER_IMAGE_YAML_DIR=/u/aceadmin
     ```

     ```
     ACE_DOCKER_IMAGE_YAML_NAME=server.conf.yaml
     ```

   - If you want to include an MQClient, download IBM MQ v9.2 or later Client deb install packages for IBM MQ on Ubuntu on z Systems from Fix Central. Place the downloaded image in the UNIX System Services file system on z/OS. Set the environment variables *ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR*, and *ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME*. For example:

     ```
     ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR=/u/aceadmin/tars
     ```

     ```
     ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME=9.2.0.0-IBM-MQC-UbuntuLinuxS390X.tar.gz
     ```

   - Specify the name of the image to build and specify a tag by setting the environment variables *ACE_IMAGE*, and *ACE_TAG*. For example:

     ```
     ACE_IMAGE=ace_jenkins_2020-06-15_09:59:01.957539
     ```

     ```
     ACE_TAG=testing
     ```

3. Choose from two methods to provide access to the running integration server. Which method you choose, depends on the level of access that you want to give to users who run IBM App Connect Enterprise commands.

   For more information, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.

4. Customize the JCL job *BIPXBLD* as described in "Customizing the JCL" on page 3048.
5. Submit the JCL job *BIPXBLD* to build the image.

What to do next

You can then use either JCL or z/OS console commands to manage your integration servers and to run IBM App Connect Enterprise commands, as described in "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL" on page 3053 and "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3056.

Loading a Docker image from a .tar file on the z/OS UNIX System Services file system

Load a Docker image from a `.tar` file on the z/OSUNIX System Services file system by running the JCL job BIPXDLI.

About this task

You can load a Docker image from a `.tar` file on the z/OS UNIX System Services file system by running the JCL job BIPXDLI.

For more information about building your own image, see "Building your own Docker image" on page 3052.

Procedure

To load a Docker image from a `.tar` archive in the z/OS UNIX System Services file system, complete the following steps:

1. Set the environment variables in the environment file, STDENV. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062. You must set the zCX access environment variables and the environment variables that are specific to Loading a Docker image from a `.tar` file on the z/OS UNIX System Services file system. Specify the name and location of the .tar file that you want to load, by setting the environment variables *ACE_DOCKER_IMAGE_TAR_DIR*, and *ACE_DOCKER_IMAGE_TAR_NAME*.
   For example:

   ```
   ACE_DOCKER_IMAGE_TAR_DIR=/u/aceadmin
   ```

   ```
   ACE_DOCKER_IMAGE_TAR_NAME=ace-s390x.tar
   ```

2. Customize the JCL job `BIPXDLI` as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
3. Submit the job `BIPXDLI`.

Results

The job `BIPXDLI` runs and the Docker image is loaded from the z/OS UNIX System Services file system into zCX, and is ready to use.

What to do next

You can then use either JCL or z/OS console commands to manage your integration servers and to run IBM App Connect Enterprise commands (with the naming convention *mqsi** or *ibmint**), as described in "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL" on page 3053 and "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3056.

Building your own Docker image

Build your own Docker image by using your own processing methods.

About this task

If you choose not to build an image by using the JCL that is supplied with IBM App Connect Enterprise on z/OS, or to use the ot4i image, you can build your own image. The image must be made available to the zCX instance, and it must be built for the s390x architecture. The following options are available:

- You can build a custom version of the ot4i image by following the instructions in the section entitled "Building a container image" in /ot4i/ace-docker.
- You can use your own image that is built without using the supplied JCL or by customizing the ot4i image. You might need to specify your own parameters to use on the docker run command that is issued when you submit the JCL jobs `BIPXIS` or `BIPXCLIS`.

Procedure

To build your own image without using the supplied JCL or by customizing the ot4i image, complete the following steps:

1. Build the image by using your own processing methods, and make the image available to the zCX instance.
   For example, place it in any Docker registry to which your zCX instance is connected. For more information, see Docker registries in the IBM z/OS documentation online, and "Loading a Docker image from a .tar file on the z/OS UNIX System Services file system" on page 3051.
2. Ensure that when you docker exec onto the container, you docker exec into an IBM App Connect Enterprise command environment. For more information, see "Setting up a command environment" on page 64.
3. Optional: If you want to administer your integration server Docker container by using JCL, complete the following steps:
   a) Set the environment variables in the environment file, STDENV. You must set the zCX access environment variables and all of the following environment variables:
      - *SERVER_NAME*. The zCX processing assumes that the Docker container is called *SERVER_NAME* and it sets *SERVER_NAME* as the container name when it starts the container.
      - *SERVER_RUN_OVERRIDE*. Specify the parameters, including port mappings to use when you submit the JCL job `BIPXIS`, which issues the docker run command.
      - *ACE_IMAGE*. You must set this environment variable for the framework to work.
      - *ACE_TAG*. You must set this environment variable for the framework to work.
   b) Submit the JCL job `BIPXIS`.
      The JCL job `BIPXIS` starts the integration server docker container by running the following command:

      ```
      docker run --name $SERVER_NAME $SERVER_RUN_OVERRIDE $ACE_IMAGE:$ACE_TAG
      ```
4. Optional: If you want to administer your integration server Docker container by using IBM z/OS console commands, you must set your integration server to write console log messages in *ibmjson* format. One way that you can set the format is to complete the following steps:
   a. **Set** *consoleLog* to **`true`**, and set *outputFormat* to **`ibmjson`** in the `server.conf.yaml` configuration file:

      ```
      Log:
                  consoleLog: true             # Control writing BIP messages to standard out.
       Set to true or false, default is true.
      ```

```
              outputFormat: 'ibmjson'      # Control the format of BIP messages written to
 standard out and file.
```

b. Set the environment variables in the environment file, BIPENVS. You must set the zCX access environment variables and all of the following environment variables:

- *SERVER_NAME*. The zCX processing assumes that the Docker container is called *SERVER_NAME* and it sets *SERVER_NAME* as the container name when it starts the container.
- *SERVER_RUN_OVERRIDE*. Specify the parameters, including port mappings to use when you submit the JCL job BIPXCLIS, which issues the docker run command.
- *ACE_IMAGE*. You must set this environment variable for the framework to work.
- *ACE_TAG*. You must set this environment variable for the framework to work.

c. Start the console listener task.

The console listener task starts the integration server Docker container by running the following command:

```
docker run --name $SERVER_NAME $SERVER_RUN_OVERRIDE $ACE_IMAGE:$ACE_TAG
```

Results

Your own image is built.

What to do next

You can then use either JCL or z/OS console commands to manage your integration servers and to run IBM App Connect Enterprise commands (with the naming convention *mqsi** or *ibmint**), as described in the following topics:

- “Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL” on page 3053
- “Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands” on page 3056

Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL

IBM App Connect Enterprise IBM z/OS Container Extensions (zCX) enable z/OS customers to run IBM App Connect Enterprise in a docker container.

About this task

When the docker image is available to zCX, you can use JCL to administer IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX).

Procedure

Complete the steps in the following tasks to manage integration servers and run IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using JCL:

- “Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL” on page 3054
- “Stopping an integration server on IBM z/OS Container Extensions (zCX) by using JCL” on page 3054
- “Deleting an integration server on IBM z/OS Container Extensions (zCX) by using JCL” on page 3055
- “Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using JCL” on page 3055

Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL

Start an integration server on IBM z/OS Container Extensions (zCX) by running the JCL job BIPXIS.

About this task

You can use your own JCL files, or the JCL files that are supplied with IBM App Connect Enterprise on z/OS.

Procedure

1. Set the environment variables in `ENVFILE`, as described in "Environment variables in STDENV and BIPENVS" on page 3062. For more information about access options, see Configuring SSH access for IBM z/OS Container Extensions (zCX).
2. Customize the JCL job `BIPXIS` as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
3. Submit the job `BIPXIS`.

Results

The job `BIPXIS` runs, and the integration server is started. The integration server log messages are written to the job log in the format that you requested; `text`, `ibmjson`, or `idText`. The log messages continue to be written when the integration server runs.

What to do next

You can deploy resources to your integration server, and run IBM App Connect Enterprise commands against it. For more information, see Chapter 7, "Deploying integration solutions," on page 2463, and "Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using JCL" on page 3055.

Stopping an integration server on IBM z/OS Container Extensions (zCX) by using JCL

Stop an integration server on IBM z/OS Container Extensions (zCX) by running the JCL job BIPXISTP.

About this task

You can use your own JCL files, or the JCL files that are supplied with IBM App Connect Enterprise on z/OS.

Procedure

1. Set the environment variables in `ENVFILE`.

   For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
2. Customize the JCL job `BIPXISTP` as advised in the job comments. By default, the integration server stops by quiescing. If it fails to stop, you can customize `BIPXISTP` to use the **`force`** option. If you want to remove the integration server Docker container, you must append the parameter **`rm`** to the `stopace.sh` command in the JCL. For more information, see "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3055, and "Customizing the JCL" on page 3048.
3. Submit the JCL job, BIPXISTP.

Results

The job `BIPXISTP` runs, and the integration server is stopped.

Deleting an integration server on IBM z/OS Container Extensions (zCX) by using JCL

Delete an integration server by running the JCL job BIPXISTP with the **rm** parameter.

About this task

You can use your own JCL files, or the JCL files that are supplied with IBM App Connect Enterprise on z/OS.

Procedure

1. Set the environment variables in ENVFILE.

   For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
2. Customize the JCL job BIPXISTP as advised in the job comments. To delete the integration server Docker container, use the `rm` command in docker by appending the parameter **rm** to the `stopace.sh` command in the JCL. If the integration server Docker container is running, it is stopped before it is deleted. For more information, see "Customizing the JCL" on page 3048.
3. Submit the JCL job, BIPXISTP.

Results

The JCL job BIPXISTP runs. The integration server is deleted.

Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using JCL

Run IBM App Connect Enterprise commands (with the naming convention *mqsi** or *ibmint**) on IBM z/OS Container Extensions (zCX) by using the JCL job, BIPXISCM.

Procedure

1. Set the environment variables in ENVFILE, as described in "Environment variables in STDENV and BIPENVS" on page 3062. For more information about access options, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.
2. Customize the JCL job BIPXISCM as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.

   For more information about the commands that are available on IBM App Connect Enterprise, see Runtime commands.

Some commands require parameters to specify the work directory of an independent integration server. For example, command - , , and systems requires you to specify **-w** *workDir*, or **--working-directory** *workDir*, depending on whether you decide to use the short names or the alternative long names for the parameters. For these commands, you must specify `/home/aceuser/ace-server` for *workDir*.

Some commands require parameters to specify the hostname or IP address of the computer on which the integration server is running, and the port on which the web user interface HTTP connection listener is listening. For example, command requires you to specify **-i** *ipAddress*, or **--admin-host** *ipAddress*, and **-p** *port*, or **--admin-port** *port*, depending on whether you decide to use the short names or the alternative long names for the parameters. For these commands, you must specify `0.0.0.0` for

ipAddress. For **-p** or **--admin-port**, you must specify the value that you set for *REST_PORT* in your environment file ENVFILE.

Some commands, for example `mqsideploy`, require one or more input files to first be made available on the docker container that runs the integration server. If the files are not available on the integration server docker container, you must use the BIPXPUT sample JCL to transfer them.

Some commands produce output. You can copy or move files from the integration server docker container by using the BIPXGET sample JCL.

BIPXPUT and BIPXGET always overwrite a file that is already there and always transfer in BINARY mode. IBM z/OS Container Extensions (zCX) run in *ASCII* mode. You can convert a file from *ASCII* to *EBCDIC* in UNIX System Services by using `iconv`. For example:

```
iconv -f ISO8859-1 -t IBM-1047 props.txt > props.ebcdic.txt
```

3. Optional: Customize the JCL job BIPXPUT as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
4. Optional: Customize the JCL job BIPXGET as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
5. Optional: Submit the JCL job BIPXPUT.
6. Submit the job BIPXISCM.
7. Optional: Submit the JCL job BIPXGET.

Results

1. If you submit the JCL job BIPXPUT, the job runs, and the input files are transferred.
2. When you submit the JCL job BIPXISCM, the job runs. The output from the IBM App Connect Enterprise command is displayed in the job log.
3. If you submit the JCL job BIPXGET, the job runs, and the output files are copied or moved.

What to do next

When you finish working with the IBM App Connect Enterprise commands, you can stop or delete the integration server. For more information, see "Stopping an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3054 and "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3055.

Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands

IBM App Connect Enterprise IBM z/OS Container Extensions (zCX) enable z/OS customers to run IBM App Connect Enterprise in a Docker container.

About this task

You can use IBM z/OS console commands to administer IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX).

Procedure

- "Copying the started task to the procedures library" on page 3057

Complete the steps in the following tasks to manage integration servers and run IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands:

- "Starting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058

- "Stopping an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058
- "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059
- "Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059

Copying the started task to the procedures library

Copy the started task to the procedures library and set the environment variables for the IBM z/OS console listener.

About this task

You can administer IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using console commands. You must set the necessary environment variables in your environment file. You must also run a supplied console listener program as a started task (one console listener program for each IBM App Connect Enterprise integration server). The console listener program is provided by the IBM z/OS Management Facility installation process, as described in "Installing and configuring IBM App Connect Enterprise on zCX" on page 3046.

Procedure

1. Optional: If you supply your own docker image, you must configure the integration server to output logs to the IBM App Connect Enterprise command line in *ibmjson* format.

   For more information, see "Building your own Docker image" on page 3052.
2. Set the environment variables that are specific to the console listener in your environment file:
   - `ACE_SCRIPTS_DIR` - set this variable to the UNIX System Services directory where the scripts are installed.
   - `SERVER_REUSE` - if you want the integration server to persist after it is stopped or if you want to reconnect to a stopped or running integration server, then set this variable to `yes`. By default, the integration server is deleted when the started task is stopped.

   For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
3. Copy the Started Task JCL into an appropriate procedures library, with the name that you agreed with the systems programmer.
   For example, copy `BIPXCLIS` into `USER.PROCLIB`. For more information, see "Planning and customization tasks on z/OS" on page 3061.
4. Customize the Started Task JCL according to the instructions in the job. For more information, see "Customizing the JCL" on page 3048.

What to do next

You can start and stop the integration server and run IBM App Connect Enterprise commands (with the naming convention *mqsi**) from the console. For more information, see the following topics:

- "Starting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058
- "Stopping an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058
- "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059
- "Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059

Starting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands

Start an integration server on IBM z/OS Container Extensions (zCX) by using the IBM z/OS console listener.

Before you begin

- Set the necessary environment variables. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
- Copy the started task JCL into an appropriate procedures library, with a name that you agreed with the systems programmer. For more information, see "Copying the started task to the procedures library" on page 3057.

Procedure

Run the console command to start the started task, which starts the integration server.
For example, if your started task is called `BIPXCLIS`, then run the following command:

```
/S BIPXCLIS
```

If the environment variable *SERVER_REUSE* is set to **yes**, the command reconnects to a running integration server Docker container or restarts a stopped one.

Results

The integration server is started. The integration server messages are written as BIP messages to ACELOG in the job log, and to the system log.

What to do next

You can deploy resources to your integration server, and run IBM App Connect Enterprise commands against it. For more information, see Chapter 7, "Deploying integration solutions," on page 2463 and "Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059.

Stopping an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands

Stop an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands.

Before you begin

- Ensure that you set the necessary environment variables. Unless *SERVER_REUSE* is set to **yes**, the console command deletes the integration server Docker container. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
- Copy the started task into the procedures library, as described in "Copying the started task to the procedures library" on page 3057.

Procedure

Run the console command to stop the started task, which stops the integration server.

For example:

```
/P BIPXCLIS
```

Results

The integration server is stopped. Unless *SERVER_REUSE* is set to **yes**, the console command deletes the integration server Docker container, and the Integration Server, as described in "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059.

What to do next

You can restart or delete the integration server by using IBM z/OS console commands. For more information, see "Starting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058, and "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3059.

Deleting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands

Delete an integration server docker container by using IBM z/OS console commands.

Before you begin

- Check the environment variables. Unless *SERVER_REUSE* is set to **yes**, the IBM z/OS console command deletes the integration server Docker container. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
- Copy the started task into the procedures library, as described in "Copying the started task to the procedures library" on page 3057.

Procedure

Run the IBM z/OS console command to stop the started task, which stops the integration server docker container.

For example:

```
/P BIPXCLIS
```

Results

- The integration server docker container is stopped.
- Unless *SERVER_REUSE* was set to **yes**, the integration server docker container is deleted.

What to do next

If you stopped the integration server docker container with *SERVER_REUSE* set to **yes**, you can delete it by using JCL. For more information, see "Deleting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3055.

Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands

Run IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands.

Before you begin

- Set the necessary environment variables. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.
- For more information about access options, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.

- Copy the started task into the procedures library, as described in "Copying the started task to the procedures library" on page 3057.

About this task

You can run IBM App Connect Enterprise commands (with the naming conventions *mqsi** or *ibmint**) on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands. The IBM App Connect Enterprise command names are case-sensitive. The commands `mqsiAssemblyInstall`, `mqsicommandconsole`, `mqsicredentials`, and `mqsivault` are not supported on IBM z/OS Container Extensions (zCX).

Some commands, for example `mqsideploy`, require one or more input files to first be made available on the docker container that runs the integration server. If the files are not available on the integration server docker container, you must use the BIPXPUT sample JCL to transfer them.

Some commands produce output. You can copy or move files from the integration server docker container by using the BIPXGET sample JCL.

BIPXPUT and BIPXGET always overwrite a file that is already there and always transfer in BINARY mode. IBM z/OS Container Extensions (zCX) run in *ASCII* mode. You can convert a file from *ASCII* to *EBCDIC* in UNIX System Services by using `iconv`. For example, if you want to convert the *ASCII* file `props.txt` to *EBCDIC*, run the following command:

```
iconv -f ISO8859-1 -t IBM-1047 props.txt > props.ebcdic.txt
```

Procedure

1. Optional: If your command requires one or more input files to be available on the docker container that runs the integration server, customize the JCL job BIPXPUT as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
2. Optional: If your command produces output, customize the JCL job BIPXGET as advised in the job comments. For more information, see "Customizing the JCL" on page 3048.
3. Optional: Submit the JCL job BIPXPUT.
4. Run the IBM z/OS console command to run an IBM App Connect Enterprise command. Specify the name of your started task JCL, the name of the IBM App Connect Enterprise command that you want to run, and any parameters that are required.
   For example, if your started task is called BIPXCLIS, and you want to run the IBM App Connect Enterprise command `mqsiexplain`, with the parameter **3750**, run the following command:

   ```
   /F BIPXCLIS,APPL='mqsiexplain 3750'
   ```

   Some commands require parameters to specify the work directory of an independent integration server. For example, command - , , and systems requires you to specify **-w** *workDir*, or **--working-directory** *workDir*, depending on whether you decide to use the short names or the alternative long names for the parameters. For these commands, you must specify `/home/aceuser/ace-server` for *workDir*.

   Some commands require parameters to specify the hostname or IP address of the computer on which the integration server is running, and the port on which the web user interface HTTP connection listener is listening. For example, command requires you to specify **-i** *ipAddress*, or **--admin-host** *ipAddress*, and **-p** *port*, or **--admin-port** *port*, depending on whether you decide to use the short names or the alternative long names for the parameters. For these commands, you must specify `0.0.0.0` for *ipAddress*. For **-p** or **--admin-port**, you must specify the value that you set for *REST_PORT* in your environment file ENVFILE.

   For more information about the commands that are available on IBM App Connect Enterprise, see Runtime commands.
5. Submit the JCL job BIPXGET.

Results

The output of the command is written to *ACECMDS* in the job log.

What to do next

When you finish running commands, you can stop the integration server by using IBM z/OS console commands. For more information, see "Stopping an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058.

Planning and customization tasks on z/OS

You must ensure that the planning and customization tasks on z/OS are completed.

Naming conventions

Decide upon a naming convention for your IBM z/OS Container Extensions (zCX) integration servers.

Each integration server must have a unique name. The Docker container in which the integration server runs, automatically uses the integration server's name as its container name. Therefore, each integration server must have a unique name for the zCX instance in which it runs.

Port allocation

Each integration server must have its own unique set of IBM App Connect Enterprise ports defined. The ports must be unique for the zCX instance on which the integration server runs:

- *REST*
- *HTTP*
- *HTTPS*
- *SSH*. For more information, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.

Environment variable definition data sets

Define data sets to hold the environment variable definitions. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.

Authorizations

Ensure that the Authorization requirements are met. The z/OS users under which the JCL or the console listener run require access to the following items:

JCL

The user ID under which the JCL runs needs:

- An OMVS segment
- A home directory
- Access to `<hlq>.SACECNTL` JCL library (read)
- Access to the UNIX System Services directory that contains the installed scripts (read + execute)
- Access to environment variable definition data sets (read)
- Access to the ssh key ZCX_SSH_KEY (read), unless you use direct access to the integration server docker container. For more information, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.

Console listener

If you want to administer IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands, you must set up the following environment for each integration server:

- Decide on the started task name for the integration server console listeners task. This name is used to set up started task authorizations, and to manage your system performance.
- Copy <hlq>.SACECNTL(BIPXCLIS) to the appropriate procedures library, for example USER.PROCLIB with the agreed name.
- Associate the started task procedure with the user ID to be used. For example, you can use the STARTED class in RACF.
- The started task user ID requires:
  - An OMVS segment
  - A home directory
  - Access to <hlq>.SACELOAD load library (read + execute)
  - Access to scripts (read + execute)
  - Access to environment variable definition data sets (read)
  - Access to the ssh key *ZCX_SSH_KEY* (read)
  - The started task writes integration server messages as BIP messages to the system log. To avoid BIP messages that have a prefix of BPXM023I (*userid*), the started task user ID also needs READ access to BPX.CONSOLE in the FACILITY class or equivalent.

Environment variables in STDENV and BIPENVS

You must set environment variables according to the task that you want to run. The values in the following sections are examples only.

The JCL and the console listener program drive shell scripts. When you submit the JCL jobs, the shell scripts are run under BPXBATCH, and the JCL job passes in an environment file that sets the environment variables. For JCL, the environment file is referred to as STDENV, and for the console listener program, the environment file is referred to as BIPENVS. For more information, see Passing environment variables to BPXBATCH.

You can put all the environment variables in one ENVFILE member or split them up into separate environment files. For example, you can have one member to specify the connection parameters, one member for build details, and one member for each integration server Docker instance for server port allocation.

The resources that you need to run IBM z/OS Container Extensions (zCX) need to be available on the z/OS UNIX System Services file system. The resources must be readable by the user ID under which the JCL or started task runs.

The values in the following sections are examples only.

Values for each zCX instance

Set the following values from the IBM z/OS Container Extensions (zCX) instance that you provision, as described in “Provisioning an IBM z/OS Container Extensions (zCX) instance” on page 3046.

- ZCX_IPADDR=9.20.2.2 - The IP address of the zCX instance to use
- ZCX_PORT=8022 - The SSH port of the zCX instance to use. Defaults to 8022 if unset. Not required for BIPXISCM, BIPXPUT, or BIPXGET if you use server ssh access. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) ” on page 3071.
- ZCX_USER=zcxuser - The user ID under which to access the zCX instance. Not required for BIPXISCM, BIPXPUT, or BIPXGET if you use server ssh access. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) ” on page 3071.
- ZCX_SSH_KEY=/u/aceadmin/.ssh/id_rsa - The SSH key to be used to access the zCX instance as ZCX_USER. Not required for BIPXISCM, BIPXPUT, or BIPXGET if you use server ssh access. For more

information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) ” on page 3071.

Values for building the Docker image

Set the following values before you submit the JCL job BIPXBLD.

- ACE_DOWNLOAD_PACKAGE_DIR=/u/aceadmin. The directory that holds the downloaded IBM App Connect Enterprise Linux on IBM Z package.
- ACE_DOWNLOAD_PACKAGE_NAME=ace-aceserver1-s390x.tar.Z. The name of the downloaded IBM App Connect Enterprise Linux on IBM Z package.
- ACE_DOCKER_IMAGE_BAR_DIR=/u/aceadmin/bars. This environment variable is optional. If it is set along with ACE_DOCKER_IMAGE_BAR_NAME, build processing includes the BAR file in the built Docker image to be made available when the integration server Docker container is run.
- ACE_DOCKER_IMAGE_BAR_NAME=*zcx.bar* . This environment variable is optional. If it is set along with ACE_DOCKER_IMAGE_BAR_DIR, build processing includes the BAR file in the built Docker image to be made available when the integration server Docker container is run.
- ACE_DOCKER_IMAGE_YAML_DIR=/u/aceadmin. This environment variable is optional. If it is set along with ACE_DOCKER_IMAGE_YAML_NAME, the integration server Docker container that uses this image uses this directory to find its server.conf.yaml configuration file.
- ACE_DOCKER_IMAGE_YAML_NAME=server.conf.yaml. This environment variable is optional. If it is set along with ACE_DOCKER_IMAGE_YAML_DIR, the integration server Docker container that uses this image uses this file as its server.conf.yaml configuration file.
- ACE_IMAGE=ace_jenkins_2020-06-15_09:59:01.957539. The name of the Docker image to be built.
- ACE_TAG=testing. The tag of the Docker image to be built. The default value is *latest*.
- SERVER_ACCESS_SSH_PUBLIC_KEY=/u/aceadmin/.ssh/aceserver1_rsa.pub. This environment variable is optional. The public key of the private/public key pair used to provide direct access to the integration server Docker container. For more information about ssh access, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) ” on page 3071.
- If you want to include the IBM MQ Client in the Docker image to be made available when the integration server Docker container is run, you must set the environment variables *ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR*, and *ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME*:
- ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR=/u/aceadmin/tars. This environment variable is optional. If you set it, you must also set ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME.
- ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME=9.2.0.0-IBM-MQC-UbuntuLinuxS390X.tar.gz. This environment variable is optional. If you set it, you must also set ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR.

Values for loading a Docker image from a .tar archive by submitting the JCL job BIPXDLI

- ACE_DOCKER_IMAGE_TAR_DIR=/u/aceadmin. The directory that contains the .tar archive file to be loaded as a Docker image. Ensure that the image to be loaded was saved by *repository*/*tag*, so that when the image is loaded, it has a repository and tag to set as *ACE_IMAGE/ACE_TAG*.
- ACE_DOCKER_IMAGE_TAR_NAME=ace-s390x.tar. The name of the .tar archive file to be loaded as a Docker image. Ensure that the image to be loaded was saved by *repository*/*tag*, so that when the image is loaded, it has a repository and tag to set as *ACE_IMAGE/ACE_TAG*.

Values for starting an integration server

- ACE_IMAGE=*ali.zcx*. The name of the Docker image to be used to run the integration server Docker container.

- ACE_TAG=*latest*. The tag of the Docker image to be used to run the integration server Docker container. The default value is *latest*.
- SERVER_NAME=*ace1*. The name of the integration server. The default value is *ace1*.
- REST_PORT=*port*. The number of the port to be used as the integration server's REST administration port. The default value is 7600. If you do not want to expose a REST administration port, set this value to 0. The value must be unique for everything that runs in the zCX instance.
- HTTP_PORT=*port*. The port number for the integration server's HTTP listener. The default value is 7800. If you do not want to expose an HTTP port, set this value to 0. The value must be unique for everything that runs in the zCX instance.
- HTTPS_PORT=*port* . The port number for the integration server's HTTPS listener. The default value is 7843. If you do not want to expose an HTTPS port, set this value to 0. The value must be unique for everything that runs in the zCX instance. Default is 7843.
- SERVER_REUSE=*yes/no*. Valid values are yes and no. The default value is no. You can set SERVER_REUSE to yes to allow processing to restart or reattach to an existing integration server container or start a new one. If you restart an integration server, the original input settings are used even if they were changed. If SERVER_REUSE is set to yes, the console listener does not attempt to remove the stopped integration server Docker container when the started task ends.

  SERVER_REUSE=yes

  When you run BIPXIS or the console listener, it checks whether an instance of the integration server container exists. One of the following events takes place:

  – If the integration server exists and is running, its logs are reattached to the new task.
  – If the integration server exists and is stopped, it is started and attached to the new task.
  – If the integration server does not exist, an attempt is made to start a new integration server container.

  When the console listener ends, it stops the integration server container but does not delete it.

  If you run BIPXIS or the console listener again, the integration server is restarted.

  SERVER_REUSE=no

  When you run BIPXIS or the console listener, an attempt is made to start a new integration server container.

  When the console listener ends, it stops the integration server container and deletes it.

  The integration server is not restarted if you run BIPXIS or the console listener.
- SERVER_ACCESS_SSH_PORT=*port*. The number of the port to be used as the integration server docker container's SSH port. The value must be unique for every docker container that runs on the zCX instance. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) ” on page 3071.
- SERVER_RUN_OVERRIDE= -p 7604:7604 -p 7804:7804 -p 7844:7843 -e LICENSE=accept. If you build your own image by using your own processing methods, specify the parameters, including port mappings to use when you issue the docker run. For more information, see “Building your own Docker image” on page 3052.

Integration server ports

If you do not want to expose a port for HTTP_PORT, HTTPS_PORT, or REST_PORT, set the value to 0.

These port numbers are for the ports that are made available outside of the integration server Docker container. For the HTTP and admin ports, to ensure that the internal port numbers that are reported in messages and statistics match these external port numbers, their internal port settings are overridden when you start the integration server. Do not set these port settings in server.conf.yaml because they are overridden. You cannot override the HTTPS port in this way. If you need to match internal and external port numbers for HTTPS, use the default value.

- SERVER_NAME. The name of the integration server. The default value is *ace1*.
- SERVER_ACCESS_SSH_PORT=*port*. This environment variable is optional. The number of the port to be used as the integration server docker container's SSH port. The value must be unique for every docker container that runs on the zCX instance.
- SERVER_ACCESS_SSH_KEY=/u/aceadmin/.ssh/aceserver1_rsa. This environment variable is optional. The private key of the private/public key pair used to provide direct access to the integration server Docker container.

For more information about ssh access, see "Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX) " on page 3071.

ACE_DEBUG

Specify this option only if you are directed to do so by your IBM service representative.

Environment variables that are addressed by STDENV and BIPENVS:

Environment variable name	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	BIPXPUT	BIPXGET	BIPXDBG	Console listener
ZCX_SSH_KEY Optional for some tasks. For more information, see Values for each zCX instance.	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Optional	Optional	Required	Required
ZCX_PORT Optional for some tasks. For more information, see Values for each zCX instance.	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Optional	Optional	Required	Required

Environment variable name	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	BIPXPUT	BIPXGET	BIPXDBG	Console listener
ZCX_USER Optional for some tasks. For more information, see Values for each zCX instance.	Required	Required	Required	Optional	Required	Required	Optional	Optional	Required	Required
ZCX_IPADDR	Required	Required	Required	Required	Required	Required	Required	Required	Required	Required
ACE_DOWNLOAD_PACKAGE_DIR	Required									
ACE_DOWNLOAD_PACKAGE_NAME	Required									
ACE_DOCKER_IMAGE_BAR_DIR	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_BAR_NAME.									
ACE_DOCKER_IMAGE_BAR_NAME	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_BAR_DIR.									

Environment variable name	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible] PUT	BIPXGET	BIPXDBG	Console listener
ACE_DOCKER_IMAGE_YAML_DIR	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_YAML_NAME.									
ACE_DOCKER_IMAGE_YAML_NAME	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_YAML_DIR.									
ACE_IMAGE	Required		Required							Required
ACE_TAG The default value is *latest*	Optional		Optional							Optional
ACE_DOCKER_IMAGE_TAR_DIR		Required								
ACE_DOCKER_IMAGE_TAR_NAME		Required								
SERVER_NAME The default value is ace1.			Optional	Optional	Optional			Optional	Optional	Optional

Environment variable name	BIPXPUT	BIPXGET	BIPXDBG	Console listener
REST_PORT. The default value is 7600. Optional for some tasks. For more information, see Integration server ports.	Optional			Optional
HTTP_PORT. The default value is 7800. For more information, see Integration server ports.	Optional			Optional
HTTPS_PORT. The default value is 7843. For more information, see Integration server ports.	Optional			Optional

Environment variable name	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	BIPXPUT	BIPXGET	BIPXDBG	Console listener
SERVER_ACCESS_SSH_PORT Optional for some tasks. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX)” on page 3071.			Optional	Optional			Optional	Optional		Optional
SERVER_ACCESS_SSH_PUBLIC_KEY Optional for some tasks. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX)” on page 3071.	Optional									

Environment variable name	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	BIPXPUT	BIPXGET	BIPXDBG	Console listener
SERVER_ACCESS_SSH_KEY Optional for some tasks. For more information, see “Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX)” on page 3071.				Optional			Optional	Optional		Optional
ACE_SCRIPTS_DIR										Required
SERVER_REUSE The default value is no.			Optional							Optional
ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME.									

Environment variable name	BIPXPUT	BIPXGET	BIPXDBG	Console listener
ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME	Optional. If you set this variable, you must also set ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR.			
SERVER_RUN_OVERRIDE	Optional			Optional
ACE_DEBUG				

Configuring access to Integration Servers running in IBM z/OS Container Extensions (zCX)

If you use the supplied processing to build a docker image, you can choose from two methods to provide access to the running integration server. Which method you choose, depends on the level of access that you want to give to users who run IBM App Connect Enterprise commands by using the supplied JCL.

About this task

Users need to access the zCX instance to interact with the integration server Docker container. Users who want to start, stop and delete integration servers, or maintain images, need to access the zCX instance by using the *ZCX_SSH_KEY*. You might prefer not to give this access to users who need only to access running integration servers and run IBM App Connect Enterprise commands. Instead, you can give those users ssh access directly into the integration server docker container without using the *ZCX_SSH_KEY*.

Procedure

Configure SSH access for IBM z/OS Container Extensions (zCX) by completing one of the following steps:

1. Optional: You can provide full access to the zCX instance by using the *ZCX_SSH_KEY*. Users who have this level of access can complete the following tasks:
   - Access a running integration server
   - Create, start, stop, and delete an integration server.
   - Maintain images.
   - Run IBM App Connect Enterprise commands.
   - Submit the JCL jobs BIPXPUT and BIPXGET

   To provide full access to the zCX instance by using the *ZCX_SSH_KEY*, complete the following steps:
   - Create an environment file ENVFILE.
   - Set the following environment variables in the environment file ENVFILE that you created:
   - – ZCX_SSH_KEY
     – ZCX_SSH_PORT
     – ZCX_SSH_USER

2. Optional: You can provide access directly into the integration server docker container without using the *ZCX_SSH_KEY*. Users who have this level of access can complete the following tasks:
   - Access a running integration server
   - Run IBM App Connect Enterprise commands.
   - Submit the JCL jobs BIPXPUT and BIPXGET

   To provide access directly into the integration server docker container without using the *ZCX_SSH_KEY*, complete the following steps:

   a. Before you create the integration server Docker image by running the JCL job BIPXBLD as described in "Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL" on page 3049, you must create an SSH key pair as described in "Provisioning an IBM z/OS Container Extensions (zCX) instance" on page 3046 but with different name. For example, run the following command:

   ```
   ssh-keygen -t rsa -b 4096 -C "your_email@domain.com" -f aceserver1_rsa
   ```

   In this example, the private key is named *aceserver1_rsa*, and the public key is named *aceserver1_rsa.pub*.

   You can create separate keys for each integration server or share keys over multiple or all integration servers, depending on your access security needs. Each key pair needs its own integration server Docker image that contains the appropriate public key.

   b. Before you create the integration server Docker image by running the JCL job BIPXBLD as described in "Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL" on page 3049, you must set the *SERVER_ACCESS_SSH_PUBLIC_KEY* environment variable. Set *SERVER_ACCESS_SSH_PUBLIC_KEY* to the name of the file that contains the public key. For example, `/u/aceadmin/.ssh/aceserver1_rsa.pub`.

   c. Before you use the image that you built in the previous step to start an integration server, as described in "Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3054, and "Starting an integration server on IBM z/OS Container Extensions (zCX) by using IBM z/OS console commands" on page 3058, you must set the environment variable *SERVER_ACCESS_SSH_PORT*. Set *SERVER_ACCESS_SSH_PORT* to specify a port number to use for ssh access to the integration server Docker container. The port number must be unique over everything that runs in that zCX instance. When the integration server Docker container starts, it also starts an ssh server that listens on the port that you specified.

   d. When you run IBM App Connect Enterprise commands by submitting the JCL job BIPXISCM, BIPXPUT, or BIPXGET, as described in "Running IBM App Connect Enterprise commands on IBM z/OS Container Extensions (zCX) by using JCL" on page 3055, do not specify *ZCX_USER* or *ZCX_SSH_KEY* in the environment variables file that is referenced by the JCL jobs.

   Set the *SERVER_ACCESS_SSH_PORT* environment variable to the port that you set when you started the integration server Docker container.

   Set the *SERVER_ACCESS_SSH_KEY* environment variable to the file that contains the private key that is associated with the public key that is built into the image. For example, `/u/aceadmin/.ssh/aceserver1_rsa`.

   If you set *SERVER_ACCESS_SSH_PORT*, only server SSH access is attempted, even if the zCX credentials are also specified. If neither are specified, the IBM App Connect Enterprise command fails.

What to do next

You can then use the supplied JCL to manage your integration servers and to run IBM App Connect Enterprise commands. For more information, see "Administering IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) by using JCL" on page 3053.

Troubleshooting on IBM z/OS Container Extensions (zCX)

Investigate and resolve errors on IBM z/OS Container Extensions (zCX).

About this task

You can use various techniques to diagnose and fix errors that occur when you run IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX).

Procedure

Environment variables

- Check the *STDOUT* and *STDERR* in the job log for errors that report missing or incorrect environment variables.

  Edit the ENVFILE to address these errors. For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.

Space

- Check the *STDOUT* and *STDERR* in the job log for error messages about lack of disk space.

  Customize and run the JCL job BIPXDSP. The job BIPXDSP removes all unused containers, unused networks, and used images (both dangling and unreferenced). For more information, see "Customizing the JCL" on page 3048.

BIPXBLD

BIPXBLD fails to unzip or untar the IBM App Connect Enterprise or MQClient packages.

- Check the *STDOUT* and *STDERR* in the job log for error messages that are related to unzipping or untarring the IBM App Connect Enterprise or MQClient packages.

  Do not to use Secure Copy Protocol (SCP) to transfer the package. SCP assumes that files are text, so binary files that are copied between EBCDIC and ASCII operating systems are corrupted during the copying process.

BIPXBLD fails to install the IBM App Connect Enterprise or the MQClient packages.

- Check the *STDOUT* and *STDERR* in the job log for error messages that are related to installing the IBM App Connect Enterprise or MQClient packages.

  Ensure that the tar file is for the *Client deb install packages for IBM MQ on Ubuntu on z Systems*, and not the *Client rpm install packages for IBM MQ on Linux for z Systems*.

Docker container issues

- Submit the JCL job BIPXDBG to obtain more information on the docker container in which the integration server is expected to run.

Console listener issues

The console listener task does not work.

- Check whether the integration server is running, by submitting the JCL job BIPXDBG, and checking the output.
  - If the integration server is stopped, cancel the console listener task.
  - If the integration is running, submit the JCL job BIPXSTP to stop it, or delete it.

The console listener fails with RC=3584.

- Check whether the user ID under which the console listener runs has execute access to the supplied scripts specified in the *ACE_SCRIPTS_DIR* environment variable.

  For more information, see "Environment variables in STDENV and BIPENVS" on page 3062.

Applying maintenance to IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX)

Apply maintenance to and install fix packs on IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX) to obtain service updates and fixes.

About this task

Service updates and other fixes are delivered occasionally in the form of APARS or fix packs. You can find information about available fix packs on the IBM App Connect Enterprise support web page.

Procedure

To apply maintenance to the IBM App Connect Enterprise package in the integration server Docker image, complete one of the following steps, depending on how you built your integration server Docker image.

- Optional: If you built your integration server Docker image by using the supplied JCL, you must complete the following steps. Run the steps on the new image that you download:
  a) Find details of the appropriate APAR or fix pack from the IBM App Connect Enterprise support web page and download it.
  b) Place the downloaded APAR or fix pack in the UNIX System Services file system on z/OS.
  c) Create a new integration server Docker image from the downloaded APAR or fix pack by using the supplied JCL, as described in “Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL” on page 3049.
  d) Start an integration server on IBM z/OS Container Extensions (zCX) to use the newly created image by running the JCL job BIPXIS, as described in “Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL” on page 3054.
- Optional: If you built your integration server Docker image by using your own processing methods, you must complete the following steps:
  a) Build your own Docker image by using your own processing methods, as described in “Building your own Docker image” on page 3052.
  b) Start an integration server on IBM z/OS Container Extensions (zCX) to use the newly created image by running the JCL job BIPXIS, as described in “Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL” on page 3054.

To apply maintenance to the MQClient libraries in the integration server Docker image, complete one of the following steps, depending on how you built your integration server Docker image:

- Optional: If you built your integration server Docker image by using the supplied JCL, you must complete the following steps. Run the steps on the new image that you download:
  a) Download IBM MQ v9.2 or later Client deb install packages for IBM MQ on Ubuntu on z Systems from Fix Central.
  b) Place the downloaded IBM MQ Client installation package in the UNIX System Services file system on z/OS.
  c) Set the environment variables *ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR*, and *ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME* to point to the newly downloaded version of the MQClient.
  For example:

```
ACE_DOCKER_IMAGE_MQCLIENT_TAR_DIR=/u/aceadmin/tars
```

```
ACE_DOCKER_IMAGE_MQCLIENT_TAR_NAME=9.2.0.0-IBM-MQC-UbuntuLinuxS390X.tar.gz
```

  d) Create a new integration server Docker image to include the new IBM MQ Client by using the supplied JCL, as described in "Creating an IBM App Connect Enterprise Integration Server Docker image on IBM z/OS Container Extensions (zCX) by using the supplied JCL" on page 3049.
  e) Start an integration server on IBM z/OS Container Extensions (zCX) to use the newly created image by running the JCL job BIPXIS, as described in "Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3054.
- Optional: If you built your integration server Docker image by using your own processing methods, you must complete the following steps.
  a) Build your own Docker image by using your own processing methods, as described in "Building your own Docker image" on page 3052.
  b) Start an integration server on IBM z/OS Container Extensions (zCX) to use the newly created image by running the JCL job BIPXIS, as described in "Starting an integration server on IBM z/OS Container Extensions (zCX) by using JCL" on page 3054.

To apply maintenance to IBM App Connect Enterprise on IBM z/OS Container Extensions (zCX), complete the following steps:

- Download the required maintenance from Fix Central and make it available on your z/OS system.
- Log on to the IBM z/OS Management Facility for your z/OS system.
- Install the maintenance by following the IBM z/OS Management Facility Software Update process described in Software Update task.

What to do next

Migration of runtime changes is not supported. For example, if you deployed a bar file to the integration server before you applied maintenance, you must redeploy it.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information (www.ibm.com/legal/copytrade.shtml).